1994

Edge effect reduction on DFT based interpolation.

Zhiwei (Jerry). Wang
University of Windsor

EDGE EFFECT REDUCTION ON DFT BASED INTERPOLATION

By

Zhiwei (Jerry) Wang

A Thesis
Submitted to the
Faculty of Graduate Studies
through the Department of
Electrical Engineering in Partial Fulfilment
of the Requirements for the Degree
of Master of Applied Science at
the University of Windsor

Windsor, Ontario, Canada

May, 1994

ISBN 0-315-93320-8

Canada

Name Zhiwei (Jerry) Wang

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics and Electrical
SUBJECT TERM

0544
SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

ABSTRACT

This thesis presents a theoretical and experimental study of discrete Fourier transform based interpolation and approximation problems. Upon the observation that the Gibbs phenomenon occurs at the vicinity of the discontinuity point of a function approximated by a Fourier series, a linear transform is introduced in this work that eliminates the discontinuity at the periodical boundary when a discrete signal with finite duration is expanded periodically due to the property of discrete Fourier transform. Hence the accuracy of the interpolation or approximation is greatly improved. This technique is also extended to two-dimensional image zooming in this thesis and much better visual results are observed. Also, a C-code program implementing the image zooming algorithm is provided in this thesis.

To my Parents

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to **Dr. J.J. Soltis** for his tremendous support and guidance throughout the progress of this research. Also, my appreciation goes to **Dr. W.C. Miller** and **Dr. G.A. Jullien** for their financial support. I would like also thank my committe members, **Dr. H.K. Kwan**, **Dr. K. Sridhar** and **Prof. P.H. Alexander**, for their suggestions and comments. My very special thanks must be given to my wife, **Winnie**, for her patience and support. Special thanks also go to my brother, **Mr. Biao Wang**, for his moral support. Last but far from the last, I would like to thank my parents, **Prof. Zhongde Wang** and **Prof. Manqin He.** Without them, none of this work could be possible.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 BASIC CONCEPT AND APPROACH

The theory of interpolation may, with certain reservations, be used to find or approximate a function f(t), from its tabulated or sampled points. In other words, if certain points of the variable $\{t_i | i=0,1,....N\}$ and the corresponding function values $\{f(t_i) | i=0,1,....N\}$ are known, and nothing further is known about the function, the problem of interpolation is to find the function values at intermediate points which are not known before[1].

This problem is obviously insoluble because any value could be inserted at an intermediate point without contradiction. It follows that some hypothesis about the general behaviour of the function must be made. First, little progress can be made unless the function is assumed to vary continuously with no sudden jump, or vary rapid variation between two given values. Secondly, it is conventional to assume that the function can be replaced, with sufficient accuracy, by a polynomial of a certain degree.

It is known that for a set of N+1 points $t_0<t_1<\cdots<t_N$ with corresponding function samples $f(t_0)$, $f(t_1)$, $\cdots$, $f(t_N)$, there always exists a unique polynomial of order N that

passes through the samples[2]. Therefore, the problem of interpolation becomes the solution of a set of linear equations as

$$f(t)=\sum_{k=0}^{N} a_k t^k \tag{1.1}$$

The set of N+1 points and their corresponding values are known and therefore the coefficients $a_0, a_1, \cdots, a_N$ are chosen so that

$$\begin{bmatrix} f(t_0) \\ f(t_1) \\ \cdot \\ \cdot \\ \cdot \\ f(t_N) \end{bmatrix} = \begin{bmatrix} 1 & t_0 & \cdots & t_0^N \\ 1 & t_1 & \cdots & t_1^N \\ \cdots & & & \\ \cdots & \cdot & & \\ \cdots & & \cdot & \\ 1 & t_N & \cdots & t_N^N \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \cdot \\ \cdot \\ \cdot \\ a_N \end{bmatrix} \tag{1.2}$$

or

$$\boldsymbol{F} = \boldsymbol{T}_N \cdot \boldsymbol{A} \tag{1.3}$$

The matrix $\mathbf{T}_N$ is a Vandermonde matrix[36] and it can be shown that

$$\det(\boldsymbol{T}_N) = \prod_{i>j} (t_i - t_j) \tag{1.4}$$

Since the points $\{t_i | i=0,1,\cdots,N\}$ are all distinct, $\det(\mathbf{T}_N)$ is nonzero, **and** $\mathbf{T}_N$ is nonsingular, the set of coefficients is therefore uniquely given by

$$\boldsymbol{A} = \boldsymbol{T}_N^{-1} \cdot \boldsymbol{F} \tag{1.5}$$

Although there is one and only one Nth-order polynomial that fits N+1 points, there are a variety of mathematical formats in which this polynomial can be expressed.

Different solutions result in different polynomials or function such as Newton's divided-difference interpolating polynomials, Lagrange interpolating polynomials, Hermite interpolating polynomials, etc. But these polynomial interpolation methods can be computationally difficult and they usually become ill-conditioned when the number of data points is large[3].

1.2 HISTORICAL SUMMARY

Historically, the mathematical process of interpolation had received widespread attention from mathematicians who were interested in the problem of tabulating (sampling) useful mathematical functions. But the question is how often a given function has to be tabulated (sampled) so that someone could use a simple interpolation rule to obtain accurate values of the function at any higher sampling rate.

In 1972, the discrete Fourier transform (DFT) was first used in interpolation[4] and thus a powerful signal analysis tool, Fourier analysis, was applied to the problem of interpolation. The computation for finding the interpolation coefficients is greatly reduced and in fact the interpolation accuracy is improved as well because of the existence of the fast algorithm for computing the Fourier transform introduced by Cooley and Tukey[5], which is now well known as the fast Fourier transform (FFT). In 1973, a digital signal processing approach to interpolation was introduced by Schafer and Rabiner[6]. In their point of view, interpolation is formulated in terms of a linear filtering operation. Actually

it shares the same theoretical concept as the fast DFT interpolation algorithm given by Prasad and Satyanarayana[7]. However, their algorithm does not generate a real sequence after interpolation since the Fourier coefficients do not have complex conjugate symmetry. Fraser proposed a new algorithm by breaking the centre coefficient (F(N/2)) into half when the number of samples is even, which generates a real signal and also increases the accuracy of the interpolation[8].

The one-dimensional interpolation was extended to two-dimensional interpolation by Sathyanarayana *et al.* in 1990[9] where again the signal after interpolation is not real. At the same time, Smit *et al.*[10] also extended the interpolation to two dimensional X-ray zooming, where a technique is developed to avoid phase shift when Skinner's FFT pruning algorithm[11] is used and gain computational efficiency. Recently, Chan *et al.* [12] proposed a two-dimensional interpolation algorithm using subsequence FFT, in which he modified the intermediate DFT coefficient sequence to keep Hermitian symmetry so a real signal is generated.

1.3 THE USE OF INTERPOLATION

Interpolation is required whenever it is necessary to increase the sampling rate from one value to another in digital signal processing. Increasing the sampling rate of an already sampled band-limited signal has many advantages such as fewer demands on A/D conversion, data transmission and storage. For example, in a speech processing system,

estimates of speech parameters are usually computed at a low sampling rate for low bit-rate storage or transmission; however, for constructing a synthesized speech signal from the low bit-rate representation, the speech parameters are normally required at much higher sampling rates[13,14]. Another example, an efficient digital realization of a frequency-multiplexed signal sub-band system is obtained by performing complicated filtering functions at a low sampling rate and simpler functions at the high sampling rate for grouping several channels into a frequency-multiplixed format[15]. In these two cases, the sampling rate must be increased by an interpolation process. Interpolation has also been applied in many other areas such as communication systems[15,16], antenna systems[17], radar systems[18], etc.

1.4 ORGANIZATION OF THESIS

This thesis consists of six chapters. The first chapter briefly introduces the basic concept of interpolation and the traditional approaches and reviews the evaluation of DFT interpolation history. Meanwhile, it also lays out the value of this work and the structure of this thesis.

In chapter two, the fundamental theory of Fourier analysis is introduced. Unlike the traditional method, the discrete Fourier transform is evaluated from the Fourier series, which is convenient for understanding of DFT-based interpolation and Gibbs phenomena reduction.

Chapter three provides both a theoretical and experimental study of the DFT based interpolation problem where the criterion for interpolation accuracy is provided. Chapter four, the major contribution of this work, explains a convergence problem of the Fourier series, which is known as the Gibbs phenomenon, at the vicinity of the discontinuity in the function. Along with other existing approaches, a new method is provided in this chapter to increase the interpolation accuracy.

In chapter five, a two-dimensional DFT interpolation algorithm is provided and is applied to image zooming problem. The two-dimensional extension of the linear sequence transform is also provided in this chapter.

The final chapter concludes this work with a summary and shows the work cited from this work, which may provide directions to future researches.

CHAPTER 2

FOURIER ANALYSIS

2.1 INTRODUCTION

In its narrowest sense, Fourier analysis or harmonic analysis of a function is a decomposition of the function into a sum of sinusoidal components[19]. In most engineering literature, Fourier analysis usually refers to either Fourier series (FS) or the discrete Fourier transform (DFT) which is often called the FFT because of the well known fast algorithm. In the past decades, the FFT is developed as an independent theory and was applied to a variety of problems. Fourier series was also normally developed independently from the Fourier integral, which is also called the Fourier transform. However, in mathematical point of view, there is no such FFT theory. The FFT is merely a fast algorithm for the numerical evaluation of the Fourier integral on a sampled waveform and the Fourier series can be derived as a special case of the Fourier integral with the introduction of distribution theory[20].

In this chapter, the underlying theory is discussed and the relationship between the Fourier integral, Fourier series, and discrete Fourier transform is addressed. It thereby

provides the framework for the development of the DFT based interpolation in chapter 3.

2.2 FOURIER TRANSFORM

It is well known that the Fourier transform of a function f(t) is defined as:

$$F(\omega)=\int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt \tag{2.1}$$

The function f(t) can also be expressed in terms of F(ω). The definition is called the inverse Fourier transform.

$$f(t)=\frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}d\omega \tag{2.2}$$

However, it is impossible to calculate the infinite integrals in Eq. (2.1) and (2.2) numerically and it is also impossible to know the value of signal f(t) with infinite length in signal processing. The above equations need to be modified for numerical evaluation.

2.3 FOURIER SERIES

A periodic function $f_p(t)$ with period T can be expressed as a Fourier series

$$f_p(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t)] \tag{2.3}$$

where ω_0 is the fundamental frequency equal to $2\pi/T$. The coefficients are given by the integral

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f_p(t) \cos(n\omega_0 t) dt \qquad n=0,1,2,... \tag{2.4}$$

and

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f_p(t) \sin(n\omega_0 t) dt \qquad n=1,2,3,... \tag{2.5}$$

By applying the identities

$$\begin{aligned} \cos(n\omega_0 t) &= \frac{1}{2}(e^{jn\omega_0 t} + e^{-jn\omega_0 t}) \\ \sin(n\omega_0 t) &= \frac{1}{2j}(e^{jn\omega_0 t} - e^{-jn\omega_0 t}) \end{aligned} \tag{2.6}$$

Eq. (2.3) can be written as

$$f_p(t) = \frac{a_0}{2} + \frac{1}{2}\sum_{n=1}^{\infty} (a_n - jb_n) e^{jn\omega_0 t} + \frac{1}{2}\sum_{n=1}^{\infty} (a_n + jb_n) e^{-jn\omega_0 t} \tag{2.7}$$

To simplify this expression, negative values of n are introduced in Eq. (2.4) and (2.5)

$$\begin{aligned} a_{-n} &= \frac{2}{T}\int_{-T/2}^{T/2} f_p(t)\cos(-n\omega_0 t)dt \\ &= \frac{2}{T}\int_{-T/2}^{T/2} f_p(t)\cos(n\omega_0 t)dt \\ &= a_n \qquad n=1,2,3,... \end{aligned} \tag{2.8}$$

and

$$\begin{aligned} b_{-n} &= \frac{2}{T}\int_{-T/2}^{T/2} f_p(t)\sin(-n\omega_0 t)dt \\ &= -\frac{2}{T}\int_{-T/2}^{T/2} f_p(t)\sin(n\omega_0 t)dt \\ &= -b_n \qquad n=1,2,3,... \end{aligned} \tag{2.9}$$

Hence

$$\begin{aligned} \sum_{n=1}^{\infty} a_n e^{-jn\omega_0 t} &= \sum_{n=-1}^{-\infty} a_n e^{jn\omega_0 t} \\ \sum_{n=1}^{\infty} jb_n e^{-jn\omega_0 t} &= -\sum_{n=-1}^{-\infty} jb_n e^{jn\omega_0 t} \end{aligned} \tag{2.10}$$

Substitution of Eq. (2.10) into Eq. (2.7) yields

$$\begin{aligned} f_p(t) &= a_0 + \frac{1}{2}\sum_{n=-\infty}^{\infty} (a_n - jb_n)e^{jn\omega_0 t} \\ &= \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \end{aligned} \tag{2.11}$$

Equation (2.11) is the Fourier series expressed in exponential form where the coefficients c_n are, in general, complex. Since

$$c_n = \frac{1}{2}(a_n - jb_n) \qquad n=0,\pm1,\pm2,... \tag{2.12}$$

the combination of Eq. (2.4), (2.5), (2.8) and (2.9) yields

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f_p(t)e^{-jn\omega_0 t}dt \qquad n=0,\pm1,\pm2,... \tag{2.13}$$

The expression for the Fourier series in exponential form as given in Eq. (2.11) with the complex coefficients given by Eq. (2.13) are normally the preferred approach in analysis.

2.4 FROM FOURIER INTEGRAL TO FOURIER SERIES

Any periodic signal $f_p(t)$ can be expressed as

$$\begin{aligned} f_p(t) &= \sum_{n=-\infty}^{\infty} f_0(t+nT) \\ &= \delta_T(t) \otimes f_0(t) \end{aligned} \tag{2.14}$$

where $\otimes$ denotes convolution. $f_0(t)$ is a function defined as

$$f_0(t) = \begin{cases} f_p(t) & 0 \le t < T \\ 0 & elswhere \end{cases} \tag{2.15}$$

$$\delta_T(t) = \sum_{n=-\infty}^{\infty} \delta(t+nT) \tag{2.16}$$

where $\delta(t)$ is the generalized function given by the following definition:

$$\delta(t) = 0 \quad for \quad t\neq 0 \quad and \quad \int_{-\infty}^{\infty} f(t)\delta(t)dt = f(0) \tag{2.17}$$

Taking the transform of both sides of Eq. (2.14), we have

$$\begin{aligned} F_p(\omega) &= \int_{-\infty}^{\infty} f_p(t)e^{-j\omega t}dt \\ &= \int_{-\infty}^{\infty} \delta_T(t) \otimes f_0(t)e^{-j\omega t}dt \end{aligned} \tag{2.18}$$

From convolution theory[21], it is known[20] that

$$\int_{-\infty}^{\infty} \delta_T(t)e^{-j\omega t}dt = \omega_0\delta_{\omega_0}(\omega) \tag{2.19}$$

where $\omega_0=2\pi/T$ and

$$\delta_{\omega_0}(\omega) = \sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_0) \tag{2.20}$$

By denoting $F_0(\omega)$ as the Fourier transform of $f_0(t)$, the Fourier transform of $f_p(t)$ therefore becomes

$$\begin{aligned} F_p(\omega) &= \omega_0\delta_{\omega_0}(\omega)F_0(\omega) \\ &= \omega_0 F_0(\omega)\sum_{n=-\infty}^{\infty} \delta(\omega - n\omega_0) \end{aligned} \tag{2.21}$$

From the property of the delta function we have

$$F_0(\omega)\delta(\omega-n\omega_0)=F_0(n\omega_0)\delta(\omega-n\omega_0) \tag{2.22}$$

therefore the Fourier transform of $f_p(t)$ is expressed as

$$F_p(\omega) = \omega_0\sum_{n=-\infty}^{\infty} F_0(n\omega_0)\delta(\omega-n\omega_0) \tag{2.23}$$

From Eq. (2.22) we can, thus, see that the Fourier transform of a periodic function $f_p(t)$ with period T consists of an infinite sequence of equidistant impulses at a distance $\omega_0=2\pi/T$ apart with amplitudes of $F_0(n\omega_0)$. Taking the inverse transform of the Eq. (2.22), we have

$$\begin{aligned} f_p(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} F_p(\omega)e^{j\omega t}d\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\omega_0\sum_{n=-\infty}^{\infty} F_0(n\omega_0)\delta(\omega-n\omega_0)e^{j\omega t}d\omega \\ &= \frac{1}{T}\sum_{n=-\infty}^{\infty} F_0(n\omega_0)\int_{-\infty}^{\infty}\delta(\omega-n\omega_0)e^{j\omega t}d\omega \\ &= \frac{1}{T}\sum_{n=-\infty}^{\infty} F_0(n\omega_0)e^{jn\omega_0 t} \end{aligned} \tag{2.24}$$

Recall that the Fourier series of a periodic function is a sum of infinite number of sinusoid in Eq. (2.11) with amplitudes given by c_n in Eq. (2.13). Compare Eq. (2.11)

with Eq. (2.23); we can see that $F_0(n\omega_0)$ is the coefficients of the Fourier series of periodic function $Tf_p(t)$. The coefficients c_n can be rewritten in the form

$$c_n = \frac{1}{T} F_0(n\omega_0) \tag{2.25}$$

Thus the coefficients as derived by means of the Fourier integral and those of the conventional Fourier series are the same for a periodic function. Except for a factor 1/T, the coefficients c_n of the Fourier series expansion of $f_p(t)$ equal the value of the Fourier transform of $f_0(t)$ at $n\omega_0$.

2.5 DISCRETE FOURIER TRANSFORM

It has been pointed out early in this chapter that the Fourier integral in Eq.(2.1) and the inversion integral in Eq.(2.2) needs to be modified so that it can be evaluated by digital computer. Consider a function f(t) with Fourier transform F(ω) as illustrated in Fig.(2.1).

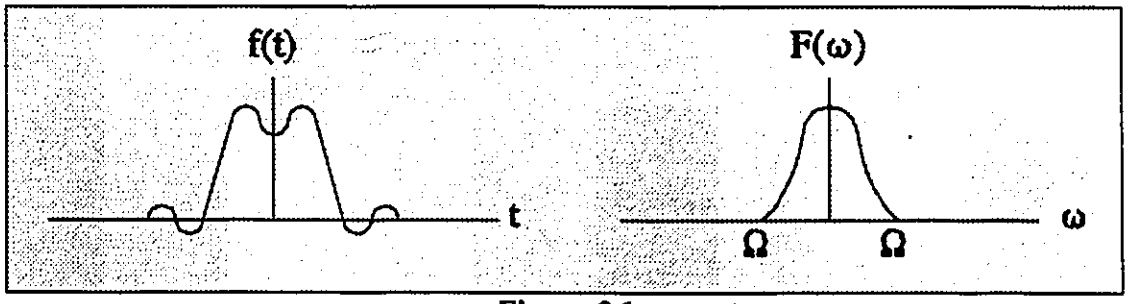

Figure 2.1
Illustration of a function f(t) and its Fourier transform

First, only a finite duration function can be considered for computer evaluation since one has finite memory. The finite duration function can be obtained by truncation of f(t) with a rectangular window w(t) defined as

$$w(t) = \begin{cases} 1 & -T/2 \le t < T/2 \\ 0 & otherwise \end{cases} \qquad (2.26)$$

where T is the duration of the window (or the duration of the function). The truncation window and its Fourier transform are illustrated in Fig.(2.2).

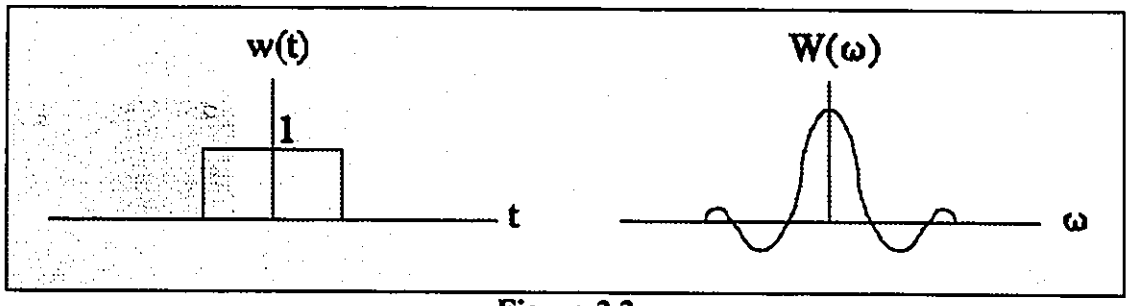

Figure 2.2
Illustration of rectangular window and its Fourier transform

The truncated function, named $f_0(t)$, can be expressed as

$$f_0(t) = f(t)w(t) = \begin{cases} f(t) & -T/2 < t < T/2 \\ 0 & otherwise \end{cases} \qquad (2.27)$$

The Fourier transform of $f_0(t)$ equals the convolution of $F(\omega)$ and the Fourier transform of the window. The finite duration function $f_0(t)$ and its Fourier transform are illustrated in Fig.(2.3).

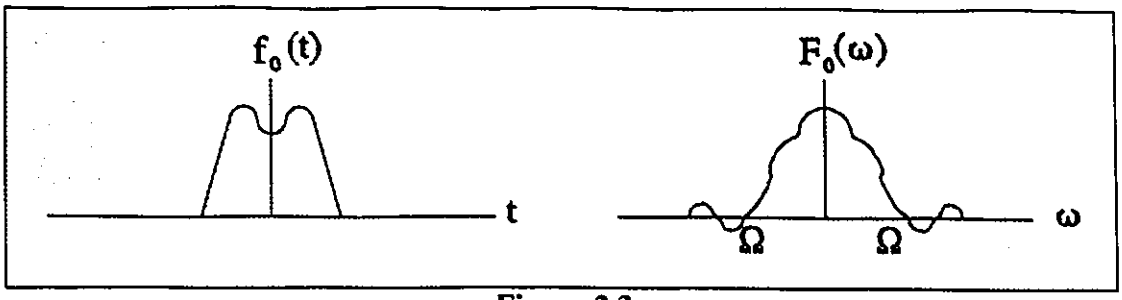

Figure 2.3
Illustration of windowed function and its Fourier transform

As illustrated in Fig.(2.3), the truncation results in the ripple or leakage in $F(\omega)$. In other words, there always exists frequency leakage or ripple in the Fourier transform of any finite duration function. It can be noticed that Eq.(2.27) is the same as the $f_0(t)$ in Eq.(2.15). When the Fourier series is used to approximate a finite duration function, the function is assumed to be equivalent to one period of a periodic function because of the periodical property of the Fourier series. The periodically expanded function corresponding to the function $f_0(t)$ in Eq.(2.27) can be expressed as

$$f_p(t) = \delta_T(t) \otimes f_0(t) \tag{2.28}$$

where $\delta_T(t)$ is defined in Eq.(2.16).

It is known from Eq.(2.22) that the Fourier transform of a periodic function is an infinite sequence of equidistant impulses at a distance $\omega_0=2\pi/T$ apart with amplitudes of $F_0(n\omega_0)$. The assumed periodic function and its Fourier transform are illustrated in Fig.(2.4).

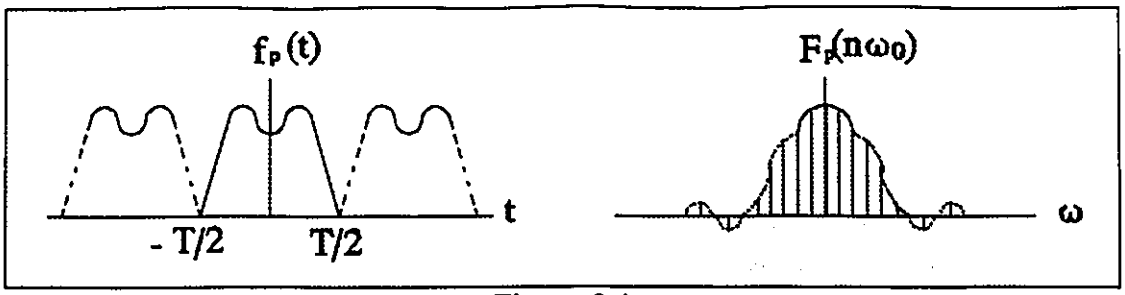

Figure 2.4
Illustration of Fourier series expression of windowed function

The periodic function can, from Eq.(2.23), be expressed as

$$f_p(t) = \frac{1}{T}\sum_{n=-\infty}^{\infty} F_0(n\omega_0)e^{jn\omega_0 t} \tag{2.29}$$

Second, the continuous function must be represented by a finite number of discrete sample values. The discrete version of the function is obtained by multiplying $f_0(t)$ with a sampling function given by

$$\delta_{T_0}(t) = \sum_{m=-\infty}^{\infty} \delta(t-mT_0) \tag{2.30}$$

and T_0 is called the sampling interval. Suppose there are N samples (N is an arbitrary integer) allocated within the duration of the function T ($T/T_0=N$). The sampled function can be expressed as

$$\begin{aligned} f_0(mT_0) &= \frac{1}{T}\sum_{n=-\infty}^{\infty} F_0(n\omega_0)e^{jn\omega_0\frac{mT}{N}} \\ &= \frac{1}{T}\sum_{n=-\infty}^{\infty} F_0(n\omega_0)e^{j2\pi\frac{nm}{N}} \qquad m=0,1,\cdots,N-1 \end{aligned} \tag{2.31}$$

By rewriting n in Eq. (2.31) as k and letting

$$k=n+rN \qquad n=0,1,\ldots,N-1 \qquad and \qquad r=\cdots,-1,0,1,\cdots \tag{2.32}$$

Eq. (2.32) can be expressed as

$$\begin{aligned} f(mT_0) &= \frac{1}{T}\sum_{n=0}^{N-1}\sum_{r=-\infty}^{\infty} F_0(n\omega_0+rN\omega_0)e^{j2\pi nm/N}e^{j2\pi mr} \\ &= \frac{1}{T}\sum_{n=0}^{N-1} e^{j2\pi nm/N}\sum_{r=-\infty}^{\infty} F_0(n\omega_0+r\omega_1) \\ & \qquad m=0,1,\cdots,N-1 \end{aligned} \tag{2.33}$$

where $\omega_1=2\pi/T_0$. It can be seen from Eq. (2.33) above that the sampling of the function results in the periodic extension (frequency aliasing) of $F_0(n\omega)$. The sampled function and its Fourier transform are illustrated in Fig.(2.5).

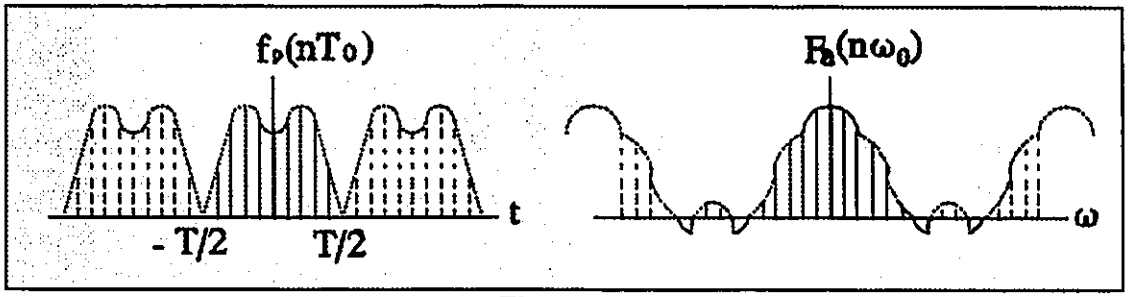

Figure 2.5
Illustration of discrete Fourier transform

Let

$$F_a(n\omega_0) = \frac{1}{T}\sum_{r=-\infty}^{\infty} F_0(n\omega_0+r\omega_1) \tag{2.34}$$

the Eq.(2.34) becomes

$$f(mT_0) = \sum_{n=0}^{N-1} F_a(n\omega_0)e^{j2\pi nm/N} \qquad m=0,1,\cdots,N-1 \tag{2.35}$$

Therefore we obtain a system of N equations whose solution yields the frequency domain samples $F_a(n\omega_0)$ in terms of the time domain samples $f(mT_0)$. The solution of Eq.(2.36) is

$$F_a(n\omega_0) = \frac{1}{N}\sum_{m=0}^{N-1} f(mT_0)e^{-j2\pi mn/N} \qquad n=0,1,\cdots,N-1 \tag{2.36}$$

The Eq.(2.37) and Eq.(2.38) are the desired discrete Fourier transform pairs. In general, $F_0(n\omega_0)$ cannot be determined in terms of $F_a(n\omega_0)$ because of the frequency leakage caused by the truncation with the rectangular window. However, in digital processing, the function f(t) is usually assumed to be a band-limit function and the leakage is assumed to be small enough to be ignored. This means

$$F(\omega) = 0 \quad for \quad |\omega| > \Omega \quad and \quad \omega_1 > 2\Omega \tag{2.37}$$

then

$$F_0(n\omega_0) = F_a(n\omega_0) \quad for \quad |\omega| < \Omega \tag{2.38}$$

Hence the solution of Eq. (2.36) yields $F_0(n\omega_0)$. If the function f(t) is not band-limited, but ω_1 is sufficiently large (which means the sampling rate $1/T_0$ is high enough) so that $F(\omega)$ can be neglected for $|\Omega|>\omega_1/2$, then $F_0(n\omega_0)$ is approximately equal to $F_a(n\omega_0)$ for $|n|<\omega_1/2\omega_0=N/2$; hence it can again be determined from Eq. (2.36).

2.6 FAST FOURIER TRANSFORM

As we have seen, within certain limits, the evaluation of the Fourier integral and Fourier series becomes the solution of a system of N equations described by Eq.(2.35). Now we consider the computational problem in the determination of the N numbers $F_a(n\omega_0)$ from Eq.(2.35). The same consideration holds for Eq.(2.36).

First, let us express the system in Eq.(2.35) as

$$F_n = \sum_{m=0}^{N-1} f_m W^{nm} \qquad n=0,1,\cdots,N-1 \tag{2.39}$$

where

$$W = e^{-j2\pi/N} \tag{2.40}$$

Clearly, N-1 additions are needed for each F_n, hence there are totally N(N-1) additions involved in Eq. (2.39). The total number of required multiplications is, also clearly, $(N-1)^2$. By employing the following technique, which is now well known as the FFT algorithm[3,22], it will be shown that the number of the multiplications can be considerably reduced.

If the given sequence f_m is expressed in terms of even and odd components as

$$a_k = f_{2k} \quad \text{and} \quad b_k = f_{2k+1} \qquad k=0,1,\cdots,[N/2] \tag{2.41}$$

then

$$F_n = \sum_{m=0}^{N-1} f_m W_N^{mn} = \sum_{k=0}^{N/2-1} f_{2k} W_N^{2kn} + \sum_{k=0}^{N/2-1} f_{2k+1} W_N^{(2k+1)n} \tag{2.42}$$

but

$$W_N^{2kn} = W_{N/2}^{kn} \qquad and \qquad W_N^{(2k+1)n} = W_{N/2}^{kn} W_N^n \tag{2.43}$$

Hence

$$\sum_{k=0}^{N/2-1} f_{2k} W_N^{2kn} = \sum_{k=0}^{N/2-1} a_k W_{N/2}^{kn} = A_n$$
$$\sum_{k=0}^{N/2-1} f_{2k+1} W_N^{(2k+1)n} = W_N^n \sum_{k=0}^{N/2-1} b_k W_{N/2}^{kn} = W_N^n B_n \tag{2.44}$$

Therefore the system in Eq.(2.41) can be expressed as

$$F_n = A_n + W_N^n B_n \qquad n=0,1,\cdots,N-1 \tag{2.45}$$

It can be noticed that the number of required multiplications to evaluate A_n or B_n equals $(N/2-1)^2$. Hence, to determine F_n from Eq.(2.42) totally $2(N/2-1)^2+N-1=N^2/2-N+1$ multiplications are needed. Compared to the number of $(N-1)^2=N^2-2N+1$ multiplications for computing F_n directly from Eq.(2.41), the use of Eq.(2.47) results in a reduction of the required number of multiplications by a factor of about 2. An additional minor reduction results from using the fact that

$$A_{n+N/2} = A_n \quad , \quad B_{n+N/2} = B_n \quad \text{and} \quad W_N^{n+N/2} = -W_N^n \tag{2.46}$$

Indeed, replacing n in Eq. (2.47) with n+N/2 and using the above equation result in

$$F_{n+N/2} = A_n - W_N^n B_n \qquad n=0,1,\cdots,N/2-1 \tag{2.47}$$

Therefore, the first half of F_n(n=0,1,....N/2-1) can be computed from Eq. (2.45) and the other half from Eq. (2.47).

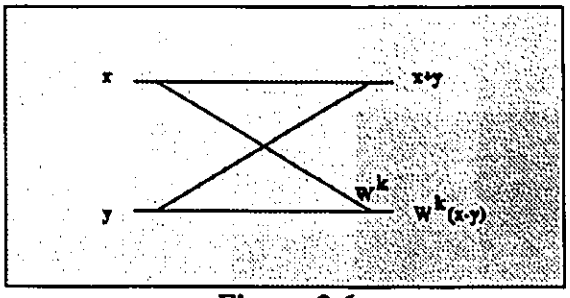

Figure 2.6
Definition of butterfly

If N/2 is also even, it can be noticed that the preceding results can be repeated; that is, A_n and B_n can be determined in terms of four DFT of order N/4. In the special case, if $N=2^s$, then the process can be repeated until the order of DFT reaches 2. By defining the signal flow diagram as shown in Fig.(2.6), the process can be represented by the flow diagram in Fig.(2.7) on the next page as a example of N=16.

It can be noticed that the starting values of f(n) in the diagram shown in Fig.(2.7) are not in the natural order. To obtain the natural order, it is necessary to employ a technique, called bit reversal. However, the FFT algorithm itself is not a major concern in this work, we only concentrate on the computing savings achieved by the FFT algorithm.

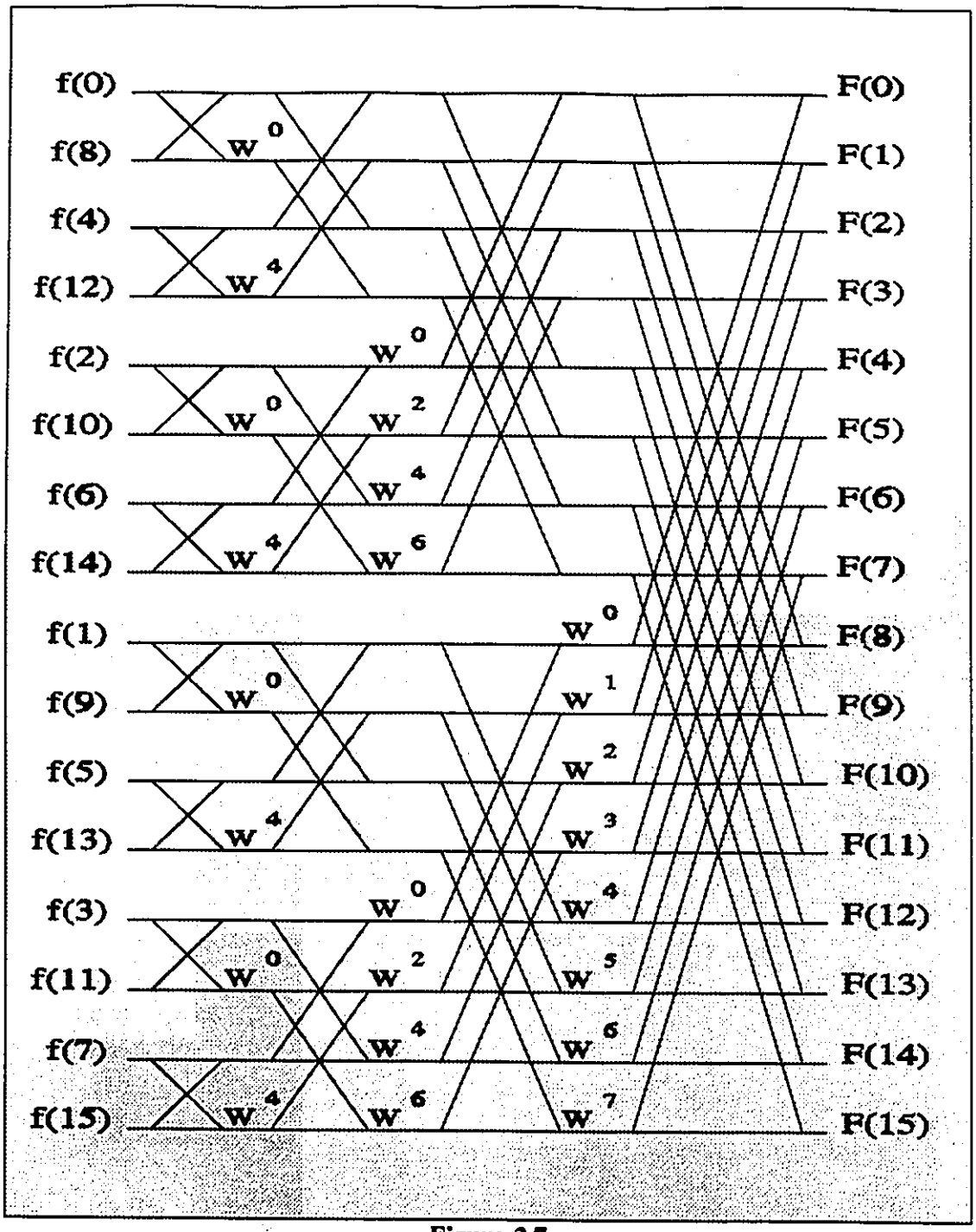

Figure 2.7
Diagram of FFT algorithm

It is easy to figure out that the required multiplications in the above system is $N/2\log_2 N$. Compared to $(N-1)^2$ required multiplications in Eq.(2.41), it is obvious that the FFT algorithm achieves remarkable reduction on the number of required multiplications for determining F_n in Eq.(2.41). Since multiplications are usually much more time consuming than additions, the reduction on the number of multiplications indeed increases the computing efficiency.

CHAPTER 3

DFT BASED INTERPOLATION

3.1. DFT AND INTERPOLATION

It has been made clear in chapter 1 that the problem of interpolation of a finite duration function can be interpreted as an approximation of the function by an independent function set (normally an orthogonal set) under the constraint that only N samples of the function f(t) are known. If the Fourier series are chosen to approximate the function f(t), the function f(t) can be expressed as

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 t} \tag{3.1}$$

where $\omega_0=2\pi/T$ (T is the function duration). It can be observed from Eq.(3.1) that because of the periodic property of the Fourier series, the function is actually periodically expanded outside the interval. The function f(t) therefore can be understood as being represented by one period of a periodic function described in Eq.(2.28). The sample value of the function can therefore be expressed as

$$f(mT_0) = \sum_{n=-\infty}^{\infty} c_n e^{jn\omega_0 mT_0} = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi nm/N} \qquad m=0,1,\cdots,N-1 \tag{3.2}$$

To obtain the polynomial approximation, replace n by k and rewrite k as

$$k = n+rN \qquad n=0,1,\cdots,N-1 \text{ and } r=\cdots,-1,0,1,\cdots \tag{3.3}$$

The Eq. (3.3) can be written as

$$f(mT_0) = \sum_{n=0}^{N-1}\sum_{r=-\infty}^{\infty} c_{n+rN} e^{j2\pi(n+rN)m/N} = \sum_{n=0}^{N-1} e^{j2\pi nm/N} \sum_{r=-\infty}^{\infty} c_{n+rN} \tag{3.4}$$

Let

$$C_n = \sum_{r=-\infty}^{\infty} c_{n+rN} \tag{3.5}$$

the Eq.(3.4) becomes

$$f(mT_0) = \sum_{n=0}^{N-1} C_n e^{j2\pi nm/N} \qquad m=0,1,\cdots,N-1 \tag{3.6}$$

The above equation is a polynomial of degree N-1 in which coefficients are C_n (n=0,1,...,N-1). Note that C_n is a combination of an infinite number of coefficients corresponding to different frequency components as shown in Eq.(3.5). This phenomenon is referred to as the aliasing of the coefficients.

Meanwhile, the Eq.(3.6) is identical in structure to Eq.(2.35). Thus the coefficients C_n can, referring to Eq.(2.36), be calculated as

$$C_n = \frac{1}{N}\sum_{m=0}^{N-1} f(mT_0)e^{-j2\pi mn/N} \qquad (3.7)$$

$$m=0,1,\cdots,N-1$$

Therefore, the fast Fourier transform algorithm can be used to increase the computation efficiency.

3.2. FAST DFT INTERPOLATION ALGORITHM

As we have seen, so far, the interpolation problem can be solved by a discrete Fourier transform pair and hence the FFT algorithm can be employed to achieve the computation efficiency. By exploring the inter-relationship of DFT coefficients and the signal samples, it has been shown that the value of interpolated samples can also be computed simply by padding zeroes to the DFT coefficients.

Let us consider a real finite duration signal sequence f(n){n=0,1,...,N-1}. (the sequence is expressed as f(n) rather than $f(nT_0)$ to make the notation simple). To use the FFT algorithm, it is necessary to assume that $N=2^s$. What we want to obtain is a length PN sequence f(n/P){n=0,1,...,PN-1}. It is obvious that the sequence f(n) provides samples for the desired sequence only at one out of each P adjacent samples under the new sampling rate. The remaining samples must be approximated by interpolation.

To interpolate those missing samples, first compute the DFT of f(n) to convert the signal to the frequency domain as

$$F(m)=\frac{1}{N}\sum_{n=0}^{N-1} f(n)e^{-jmn\frac{2\pi}{N}} \qquad m=0,1,\ldots,N/2-1 \tag{3.8}$$

The overall scale factor 1/N is included in the forward rather than in the inverse transform to ensure that the scale of F(m) is independent of N. The second half of the DFT coefficients for m=N/2+1,....N-1, is redundant for real data because N/2 is the folding frequency or Nyquist limit.

Next construct a new sequence G(m) of complex coefficients of length PN by padding zeroes to F(m) as

$$G(m) = \begin{cases} F(m) & m=0,1,\ldots,N/2-1 \\ 0.5F(m) & m=N/2 \\ 0 & m=N/2+1,N/2+2,\ldots,PN/2 \end{cases} \tag{3.9}$$

The original data are real, therefore the DFT coefficients exhibit conjugate symmetry, and the remainder of the full-length sequence can be obtained from eq. (3.8) according to

$$G(M-m)=G(m)^* \qquad m=1,2,\ldots,PN/2 \tag{3.10}$$

where * denotes complex conjugation.

Note that all terms in Eq. (3.9) are carried over to the new sequence with equal weighting except the term at the original folding frequency m=N/2, which is given a relative weighting of one half. The reason is as follows. The new folding frequency of the DFT is PN/2, about which the conjugate symmetry of Eq. (3.10) pivots. So the term at frequency N/2 is mirrored by Eq. (3.10) to PN-N/2. Thus, whereas this term is summed only once in an inverse DFT of length N, it is summed twice in the inverse DFT of length PN (other terms are summed twice in both cases). Therefore, it is necessary to give a weight of one half to this coefficient.

The last step is to perform the inverse DFT of the sequence G(m). Since G(m) (m=0,1,....,PN-1) keeps the complex conjugate symmetry, the resulting sequence, say g(n), is a real value sequence of length PN.

$$g(n)=\sum_{m=0}^{PN-1} G(m)e^{jmn\frac{2\pi}{PN}} \qquad n=0,1,...,PN-1 \tag{3.11}$$

To prove the obtained sequence g(n) is the interpolated sequence from f(n), we must prove that g(n) possesses the property of g(Pn)=f(n) (n=0,1,....,N-1). The original sequence f(n) can be obtained from the DFT coefficients F(m) by

$$f(n)=\sum_{m=0}^{N/2-1} F(m)e^{j\frac{2\pi}{N}mn}+F(N/2)e^{j\frac{2\pi}{N}\frac{N}{2}n}+\sum_{m=1}^{N/2-1} F^{*}(m)e^{-j\frac{2\pi}{N}mn} \tag{3.12}$$

$$n=0,1,\cdots,N-1$$

Substituting Eq. (3.9) and Eq. (3.10) in Eq. (3.11) we have

$$\begin{aligned} g(n) &= \sum_{m=0}^{M/2} G(m)e^{j\frac{2\pi}{PN}mn} + \sum_{m=1}^{M/2-1} G^*(m)e^{-j\frac{2\pi}{PN}mn} \\ &= \sum_{m=0}^{N/2-1} F(m)e^{j\frac{2\pi}{PN}mn} + \frac{1}{2}F(\frac{N}{2})e^{j\frac{2\pi}{PN}\frac{N}{2}n} \\ &\quad + \frac{1}{2}F^*(\frac{N}{2})e^{-j\frac{2\pi}{PN}\frac{N}{2}n} + \sum_{m=1}^{N/2-1} F^*(m)e^{-j\frac{2\pi}{PN}mn} \end{aligned} \tag{3.13}$$

$$n=0,1,\cdots,PN-1$$

Substituting n as Pn in the above equation results in

$$\begin{aligned} g(Pn) &= \sum_{m=0}^{N/2-1} F(m)e^{jmn\frac{2\pi}{N}} + \frac{1}{2}F(\frac{N}{2})e^{j\frac{2\pi}{N}\frac{N}{2}n} \\ &\quad + \frac{1}{2}F^*(\frac{N}{2})e^{-j\frac{2\pi}{N}\frac{N}{2}n} + \sum_{m=1}^{N/2-1} F^*(m)e^{-j\frac{2\pi}{N}mn} \end{aligned} \tag{3.14}$$

$$n=0,1,\cdots,N-1$$

since $e^{jn\pi}=e^{-jn\pi}$ and F(N/2)=F*(N/2), we hence obtain g(P·i)=f(i) {i=0,1,2,...,N-1}. Therefore, the sequence g(n) is the desired interpolated sequence.

3.3 INTERPOLATION IN THE FREQUENCY DOMAIN

It has been shown, so far, that DFT based interpolation is a process of converting the signal into the frequency domain, padding zeroes to the high frequency part and reconstructing the new signal. In this section, we will explain how this works from the

digital signal processing point of view.

Consider a continuous-time signal f(t) with Fourier transform

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt \tag{3.15}$$

The signal f(t) is sampled to produce the sequence $f(n)=f(nT_0)$ where T_0 is the sampling period. The z transform of the sequence f(n) is defined as

$$F(z) = \sum_{n=-\infty}^{\infty} f(n) z^{-n} \tag{3.16}$$

The z transform evaluated on the unit circle $F(e^{j\omega T})$ is called the Fourier transform of the sequence f(n). It is well known that the Fourier transform of the sequence f(n) is related to the Fourier transform of f(t) by[23]

$$F(e^{j\omega T}) = \frac{1}{T_0} \sum_{k=-\infty}^{\infty} F(\omega + k\frac{2\pi}{T_0}) \tag{3.17}$$

If f(t) is band-limited, i.e., $F(\omega)=0$ for $|\omega| > \pi/T_0$, then it can be seen from Eq. (3.17) that

$$F(e^{j\omega T}) = \frac{1}{T_0} F(\omega) \qquad -\frac{\pi}{T_0} \le \omega \le \frac{\pi}{T_0} \tag{3.18}$$

If the sampling rate is increased by an integer factor P, then the new sampling rate is $T'_0=T_0/P$. This simply implies that the new sequence is

$$\begin{aligned} g(n) &= f(nT') \\ &= f(nT/P) \\ &= f(n/P) \end{aligned} \tag{3.19}$$

This means that the sequence f(n) provides only one of every P samples of the desired sequence at the new sampling rate. The remaining samples must be filled in by interpolation.

Consider a new sequence

$$f'(n) = \begin{cases} f(n/P) & n=0,\pm P,\pm 2P,\ldots \\ 0 & otherwise \end{cases} \tag{3.20}$$

The z transform of this sequence is

$$\begin{aligned} F'(z) &= \sum_{n=-\infty}^{\infty} f(n/P)z^{-n} \\ &= \sum_{n=-\infty}^{\infty} f(n)z^{-Pn} \\ &= F(z^{P}) \end{aligned} \tag{3.21}$$

The Fourier transform of f'(n) therefore is

$$\begin{aligned} F'(e^{j\omega T'}) &= F(e^{j\omega T'P}) \\ &= F(e^{j\omega T}) \end{aligned} \tag{3.22}$$

Thus $F'(e^{jwT'})$ is periodic with $2\pi/T_0=2\pi/PT'_0$, rather than $2\pi/T'_0$ as generally used for sequences associated with a sampling period T'_0.

If we wish to obtain the sequence g(n) from f(n), which is supposed to be obtained from sampling function f(t) at new sampling rate T'_0, then it is must be ensured that

$$G(e^{j\omega T'})=\frac{1}{T'}F(\omega) \qquad -\frac{\pi}{T'}\leq\omega\leq\frac{\pi}{T'} \tag{3.23}$$

Comparing $G(e^{j\omega T'})$ with $F(e^{j\omega T})$ in Eq.(3,17), it is clear that the images of $(1/T)F(\omega)$ in $F'(e^{j\omega T'})$ which are centred from $\omega=2\pi/T$ to $2(P-1)\pi/T$ must be removed by a digital low-pass filter that rejects all frequencies in the range $\pi/T<|\omega|<\pi/T'$. Furthermore, to ensure that the amplitude is correct for sampling interval T', the gain of the filter must be P. That is

$$\begin{aligned} G(e^{j\omega T'}) &= H(e^{j\omega T'})F'(e^{j\omega T'}) \\ &= H(e^{j\omega T'})F(e^{j\omega T}) \\ &= \frac{1}{T}H(e^{j\omega T'})F(\omega) \end{aligned} \tag{3.24}$$

where $H(e^{j\omega T'})$ is periodic with $2\pi/T'$ and

$$H(e^{j\omega T'}) = \begin{cases} P & |\omega|\leq\pi/T \\ 0 & \pi/T<|\omega|<\pi/T' \end{cases} \tag{3.25}$$

From the above discussion, it can be seen that the interpolation scheme requires

two steps: first zero packing the sequence f(n) by inserting P-1 zero-valued samples between each value of the original sequence, and secondly, perform a low-pass filtering that removes the redundant frequency from $F(e^{j\omega T})$. Therefore, the zero-padding in the frequency domain, which is the discrete Fourier transform based interpolation scheme, is equivalent to zero-packing the original sequence and performing an ideal low-pass filtering at the same time[24].

3.4 EXPERIMENTAL STUDY

In the numerical experiments, a test signal f(t) with a finite duration T is sampled at sampling rate f_s to obtain f(nT) {n=0,1,....,N}. Then the signal is interpolated or reconstructed at a higher sampling rate f'_s. The $f'_s=P \cdot f_s$ where P must be an integer of power of two as mentioned before. The reconstructed discrete signal g(nT') is compared with the original function at the sampling rate of f'_s covering same period, i.e., f(nT/P).

In digital signal processing application, signals are, in general, band limited or viewed as band limited. Such signals can be understood as the superposition of sinusoids within a frequency band. For this reason, a cosine function is used as the test signal in our experiments. The test signal is given as

$$f(t)=\cos(2\pi ft+\phi) \qquad 0\le t<T \tag{3.26}$$

In our experiments, the sampling rate f_s=5512.5(Hz) and P=8. Therefore f'_s is 44.1k(Hz). The reason is that the sampling rate of 44.1k(Hz) is the typical sampling rate

for obtaining a high quality speech signal. (The band of a telephone quality speech signal is 300-3000 Hz). To limit the computation time for long signals and because of the length limitation for a digital computer, the number of samples N_s under sampling rate f_s is chosen to be 128, which means the time duration of signal f(t) is $T=N_s/f_s=0.02s$. For longer signals, we will divide the signal into several frames and interpolate from one frame to another. This technique will also be discussed later in experiments.

To examine the reconstruction accuracy, several criteria are selected. They are absolute error (AE_n) and normalized mean square error (NMSE). These criteria are suitable for examining the reconstruction accuracy but from different points of view. The NMSE emphasizes the overall accuracy and the AE_n is the individual error for each point. They are defined as below:

$$AE_n=|g(nT')-f(nT')| \qquad n=0,1,...,PN_s-1 \tag{3.27}$$

$$NMSE=10\log\frac{\sum_{n=0}^{PN_s-1}[g^2(nT')-f^2(nT')]}{\sum_{n=0}^{PN_s-1}f^2(nT')} \tag{3.28}$$

Figure 3.1 shows the AE curve for $f_s=1500$(Hz), $\phi=0$. The curve shows that the error occurs between every two "perfect" interpolated samples and it has the trend that the AE error reached it's minimum at the centre and increases very slowly through the

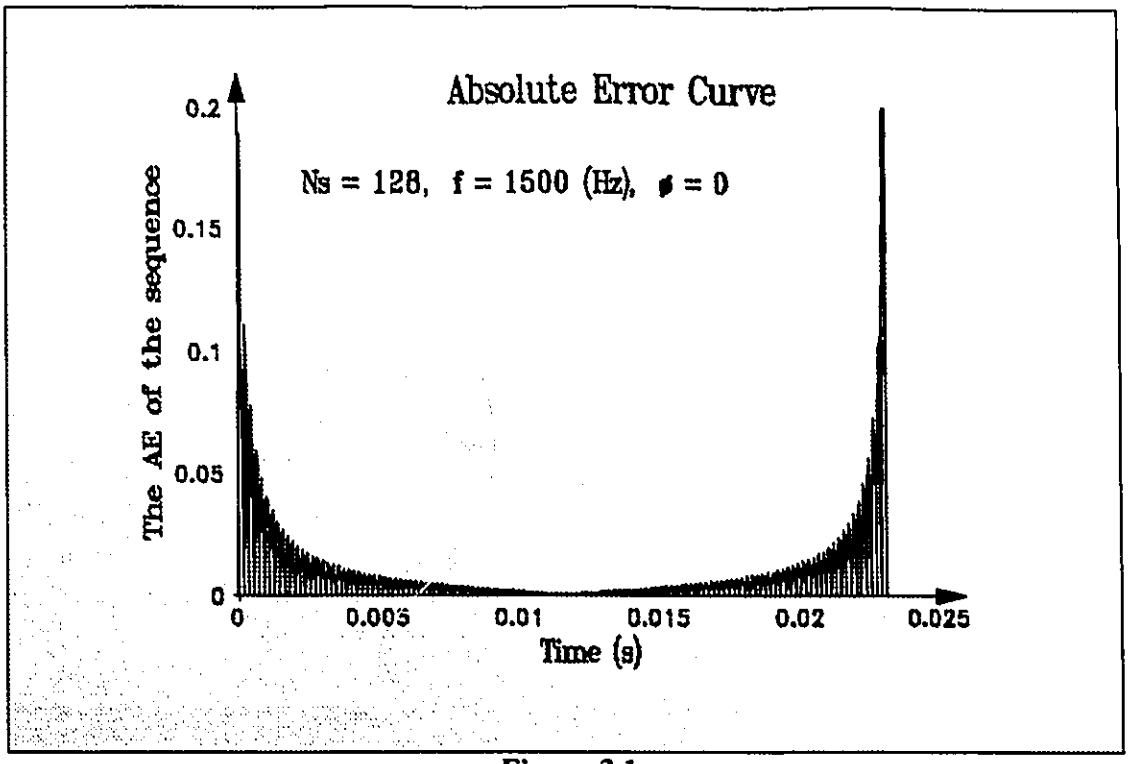

Figure 3.1

centre half part of the sequence and climbs up steeply to the maximum at each side of the sequence. As we mentioned in chapter 2, this is also the reason that the interpolation by the DFT is only an approximation no matter how small the error is and how it can be further reduced.

In order to simplify the statistics and to make the comparison easy, the AE curve is smoothed by taking the average value over a rectangular window and passing the window over the sequence. The width of the window is equal to the distance between samples of perfect interpolation. This choice gives good smoothing results while preserving the important local trends as shown as the solid curve in Figure 3.1.

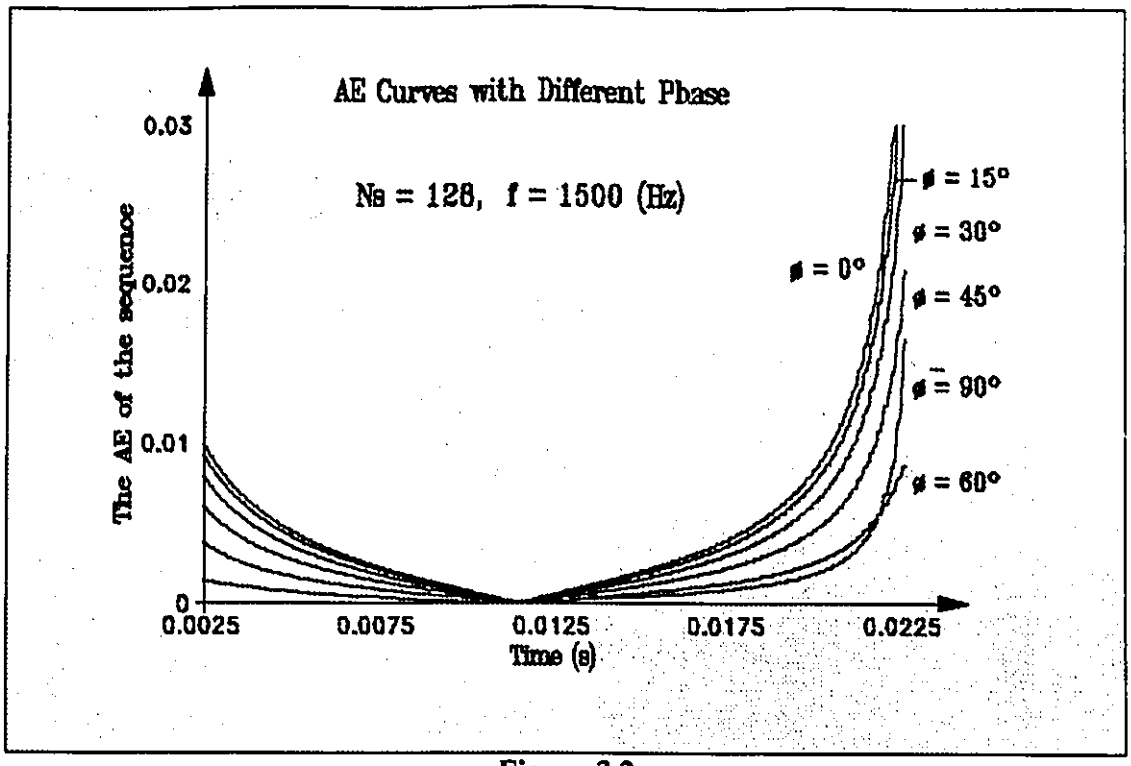

Figure 3.2

Figure 3.2 shows a family of smoothed AE curves for cosine waves for different phase shifts. For the convenience of comparison, the curves are displayed showing about 85% near the centre of the sequence. All curves have the same characteristic feature, having the minimum at the centre of the sequence and increasing along each side. We notice that certain phase shift results in lower error, which is called the fortuitous end-around effect by Fraser[8]. The reason behind will becomes clear in next chapter where the Gibbs phenomenon is discussed.

To simplify the computation, the phase of the testing signal is set to zero henceforth unless otherwise declared. Fig. (3.3) shows the NMSE curve as a function of

frequency (f ranges from 300-3000 Hz). It is observed again that "perfect" interpolation occurs at some frequencies. "Failed interpolation" (0dB) or above occurs for $f>f_0+f_s/2=3056$ (Hz), which is the Nyquist limit. This chart graphically illustrates the sampling theorem in practice. The Nyquist limit is an absolute barrier to successful interpolation or reconstruction.

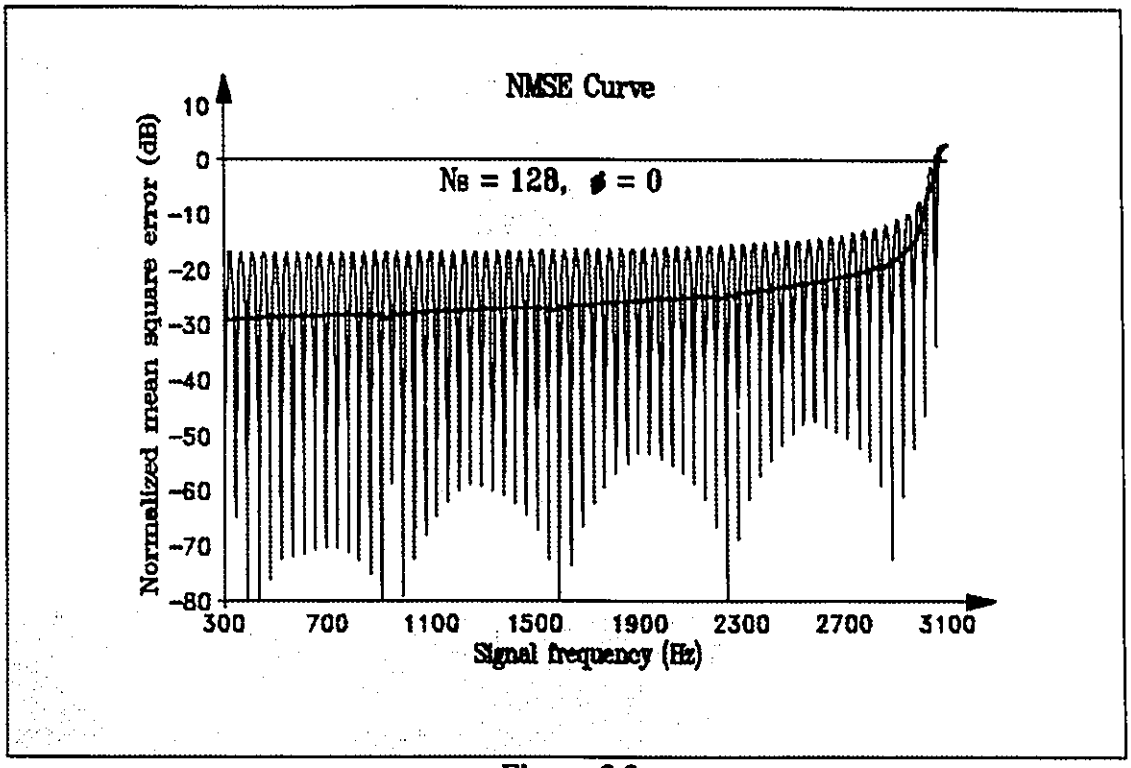

Figure 3.3

Just as for Figure 3.1, the NMSE curve is smoothed using the window technique, i.e., taking the average within the window and passing the window through all frequencies. The width of the window here is also the distance between the "perfect" interpolation frequencies. The smoothed curve is shown as the solid line in the plot. It can

be noticed that the NMSE is inversely proportional to the signal frequency. But for f<2700(Hz), the change is very small and the region could be considered as a plateau or stable region. As the signal approaches the Nyquist limit, the NMSE increases rapidly and reaches the "failed" interpolation error -- 0dB at the Nyquist limit.

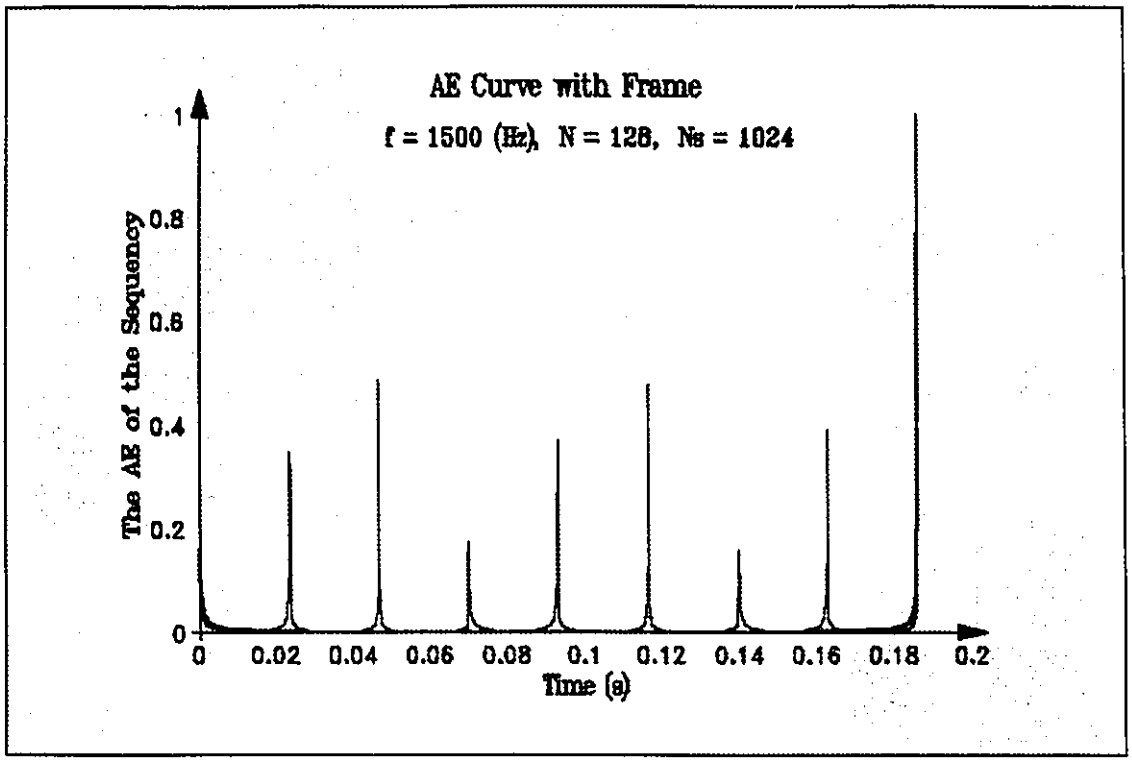

Figure 3.4

As mentioned at the beginning of this section, a long duration signal could be divided into several short period frames and dealt with one frame at a time. To illustrate this, the same signal with a longer time duration is chosen as the test signal. In the experiments, the total number of points is chosen as N_s=1024, which means the time duration $T=N_s/f_s$=0.16s. The frame size is chosen as N=128 same as the signal length in

previous experiments. Figure 3.4 shows the AE curve throughout the whole signal period. The peaks occur at the edge of frames due to the interpolation error. The peak value is about 80% of the signal maximum amplitude which is rather high. For comparison, the solid-line curve shows the AE curve for taking the whole signal period in one frame. The major difference, as we can see, is those peaks occurred at the edges of the frame in the centre of the sequence.

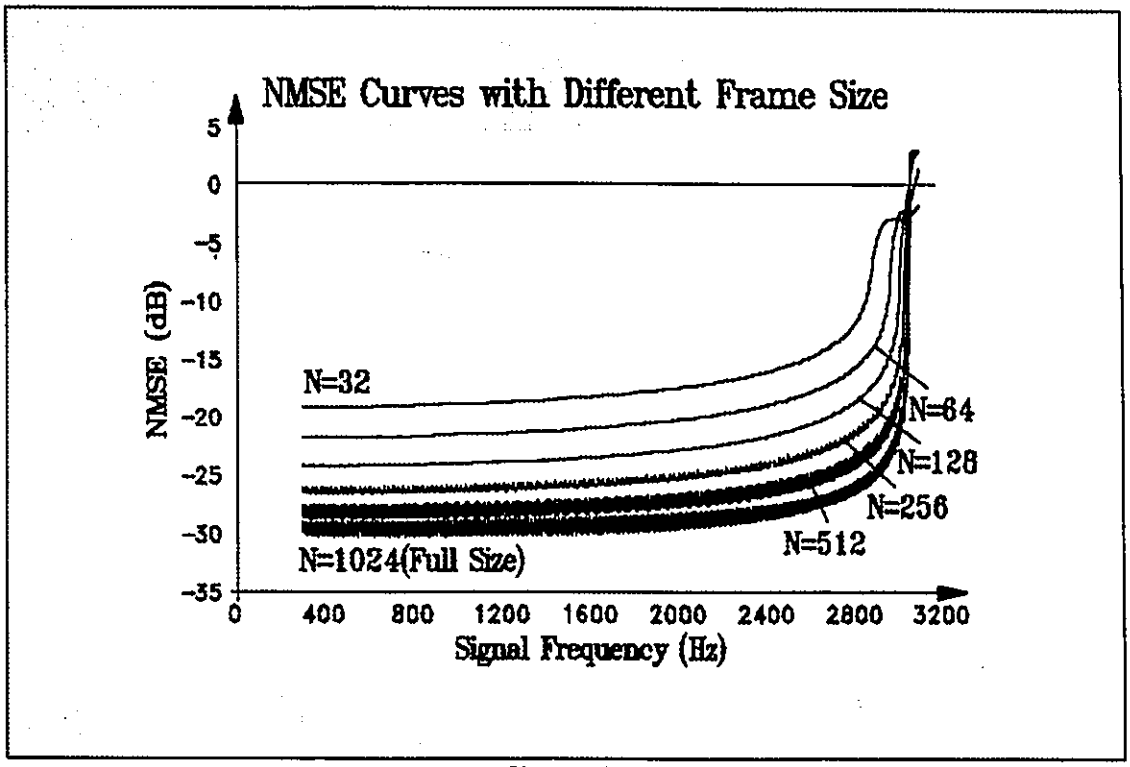

Figure 3.5

Figure 3.5 shows a family of smoothed NMSE curves with different frame lengths. The curves have a characteristic shape same as in Fig.(3.3). The NMSE is approximately unchanged for f<2700(Hz) and it is inversely proportional to the length of the frame. The

reason is obvious that more peaks occur for shorter frames. It will be shown in the next chapter that the frame edge peak can be reduced by proper techniques.

CHAPTER 4

EDGE EFFECT REDUCTION

4.1 GIBBS PHENOMENON

The Gibbs phenomenon is named because of a letter Gibbs wrote to Nature in 1899[25] in which he discussed the poor convergence of Fourier series in the vicinity of the jump of the sawtooth function.[1] However, his statement was not accompanied by any proof and this remarkable observation passed practically unnoticed for several years. In 1906, Bocher returned to the subject in a memoir[28] on Fourier series and greatly extended Gibbs's result. He showed, among other things, that the phenomenon which Gibbs has observed in the case of a particular Fourier series (i.e., Fourier series for sawtooth function) holds in general at ordinary points of discontinuity. To quote his own words:

If f(x) has the period 2π and in any finite interval has no discontinuities other than a finite number of finite jumps, and if it has a derivative which in any finite interval has no discontinuities other than a finite number of finite discontinuities, then as N becomes

1 Actually Gibbs phenomenon was first described by the British mathematician Wilbraham in 1848[26]; see Carslaw[27] for the history.

infinite the approximation curve produced by Fourier series approaches uniformly the continuous curve made up of

(a) the discontinuous curve f(x).

(b) an infinite number of straight lines of finite lengths parallel to the y-axis and passing through the points $a_1, a_2, \ldots$ on the x-axis where the discontinuities of f(x) occur. If a is any one of these points, the line in question extends between the two points whose ordinates are

$$f(a-0)+\frac{DP_1}{\pi} \quad , \quad f(a+0)-\frac{DP_1}{\pi} \tag{4.1}$$

where D is the magnitude of the jump in f(x) at a, i.e., D=f(a+0)-f(a-0), and

$$P_1=\int_{\pi}^{\infty}\frac{\sin x}{x}dx=-0.2811 \tag{4.2}$$

As mentioned in chapter 2, when the discrete Fourier transform is performed on a finite duration signal, it is assumed that the signal is described within one period of a periodic signal as

$$\begin{aligned} f_p(t) &= f(t)\otimes\delta_T(t) \\ &= \sum_{n=-\infty}^{\infty} f(t+nT) \end{aligned} \tag{4.3}$$

If $f(-T/2)\neq f(T/2)$, then the periodical function $f_p(t)$ has a discontinuities at the periodic boundaries. Therefore the Gibbs phenomenon occurs in the vicinity of the periodic

boundary when the function is reconstructed from its discrete Fourier coefficients[1]. However, the proof of the Gibbs phenomenon is beyond the scope of this thesis. Interested readers can refer to Carslaw[29] and Dym[30].

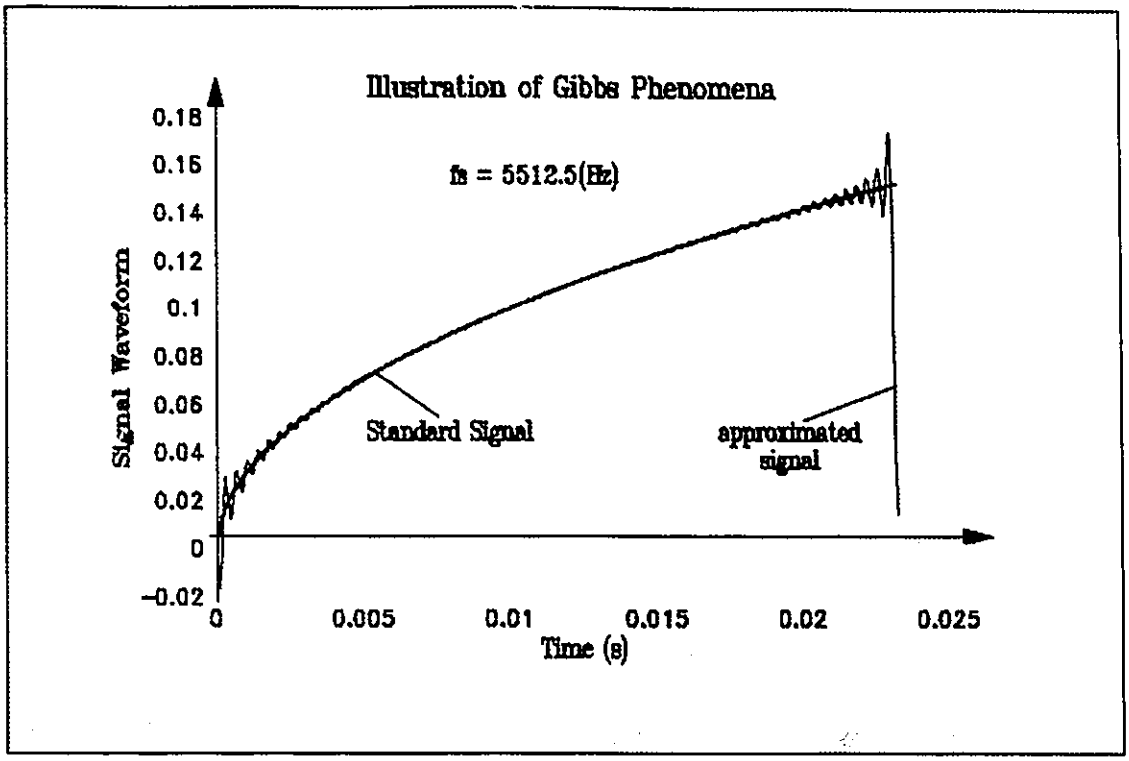

Figure 4.1

Fig.(4.1) shows a square root function curve with a finite duration and its reconstructed or interpolated function curve generated from its Fourier coefficients given by Eq.(3.7)-(3.10). The figure illustrates the Gibbs phenomenon clearly in the vicinity of the periodic boundary at the ends of the signal. This is why the interpolation error tilts

1 As shown in chapter 2, the DFT coefficients is the aliased coefficients of Fourier series.

up at the sequence ends and the reason why the peaks occur at the edge of each frame in the absolute error curve.

4.2. LEAKAGE REDUCTION

Recall that frequency leakage is inherent in the discrete Fourier transform because of the time domain truncation by the rectangular window mentioned in chapter 2. In general, the truncation results in a sharp discontinuity in the periodic function which is expanded from the truncated function as shown in Eq.(4.3). The discontinuity, in other words, results in side-lobes in the frequency domain, which are termed frequency leakage or ripples and which is the major reason for the interpolation error as illustrated in Fig. (4.1).

To eliminate the discontinuity, Fraser[8] proposed a window technique that makes the function continuous at the periodic boundary. Consider a function f(t) with finite duration of $0 \le t < T$. Rather than being directly approximated by the coefficients of its discrete Fourier transform, the function f(t) is first modified by multiplying it by a so-called half cosine-bell window with the same duration to generate a new function, $f_w(t)$, and approximating the result by the coefficients of the discrete Fourier transform on the modified function. The modified function, named $f_w(t)$, is given by

$$f_w(t) = f(t) \cdot w(t) \qquad 0 \le t < T \tag{4.4}$$

and the half cosine-bell window w(t) is given by

$$w(t) = \begin{cases} 0.5[1-\cos(2\pi t/T_w)] & 0 \le t < T_w/2 \\ 0.5[1-\cos(2\pi(T-t)/T_w)] & T-T_w/2 < t < T \\ 1 & otherwise \end{cases} \quad (4.5)$$

where $T_w<T$ is the combined length of windowed region. Fig.(4.2) illustrates the waveform of such a window with $T_w=T/2$. If $T_w=T$, a cosine bell extends over the full signal period, corresponding to a Hanning window.

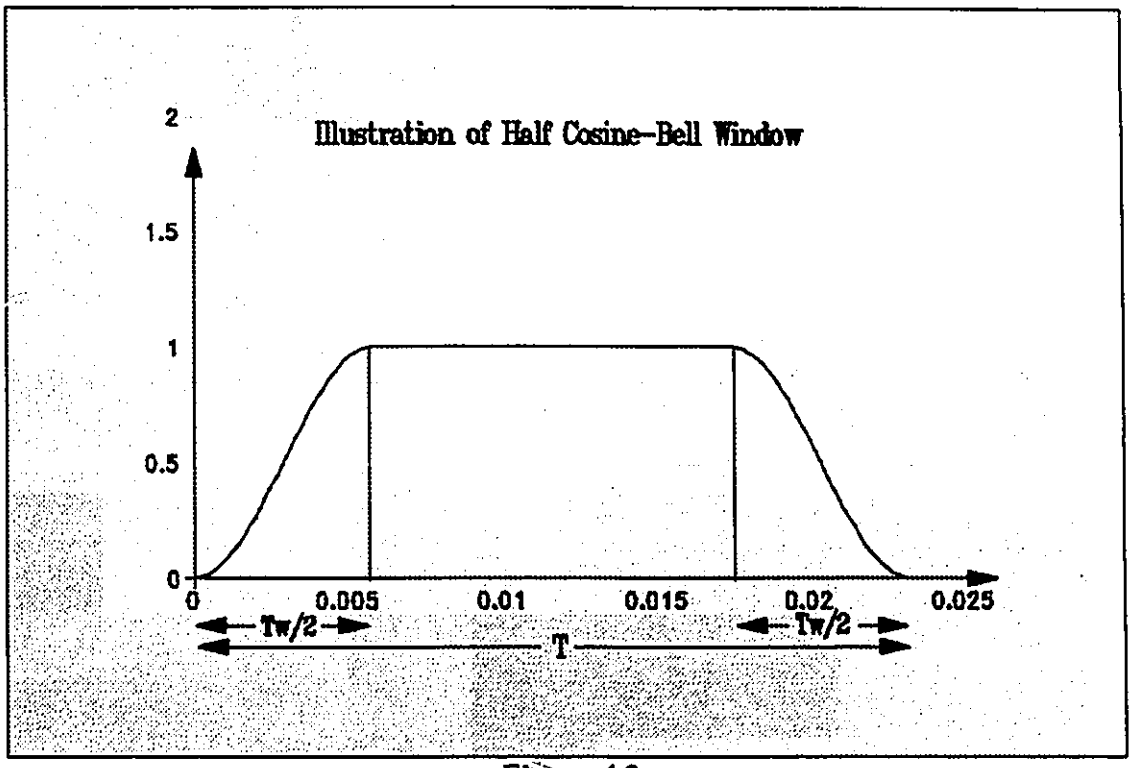

Figure 4.2

As can be seen from Eq.(4.5), the window function possesses the property that w(0)=w(T)=0 which means the modified function also has $f_w(0)=f_w(T)=0$. Therefore there

is no discontinuity on the periodic function expanded from the modified function and hence the Gibbs phenomenon, or the oscillations in the vicinity of discontinuity will be reduced since the amplitude of the maximum ripple corresponds to the height of the jump of the discontinuity as shown in Eq.(4.1). To illustrate this, Fig.(4.3) shows the modified function from the square root function shown in Fig.(4.1).

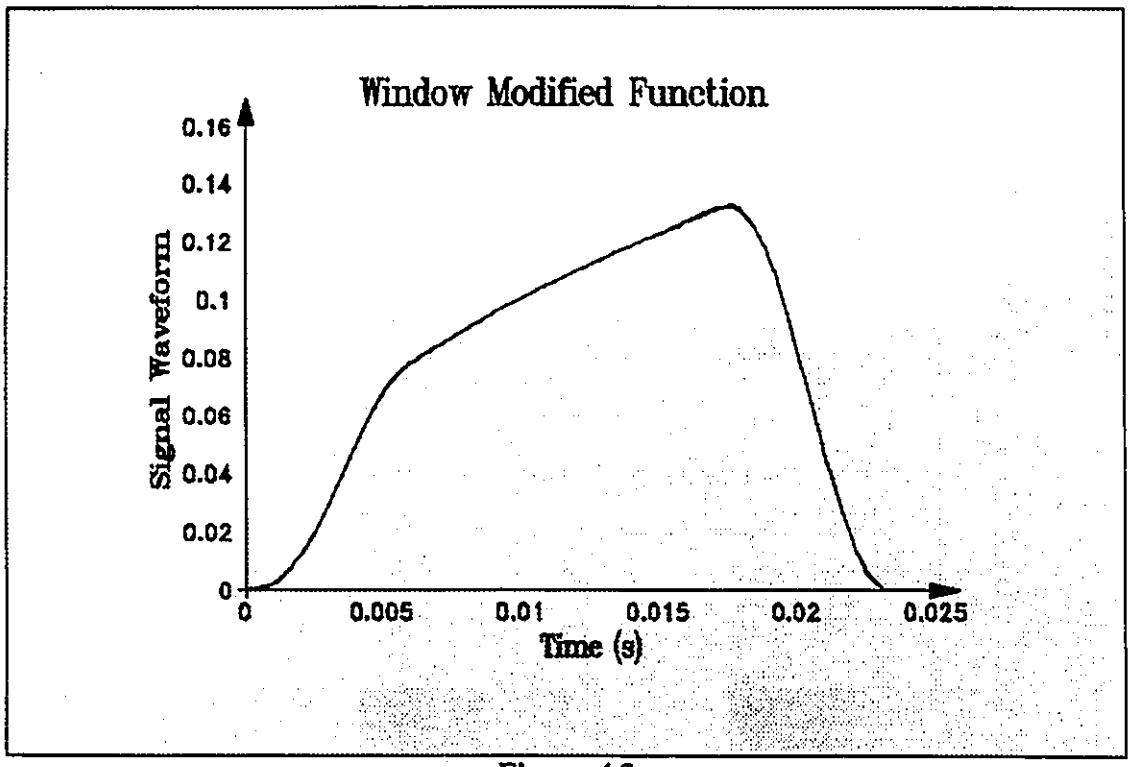

Figure 4.3

The waveform reconstructed from the coefficients of its discrete Fourier transform is also illustrated in Fig.(4.3) as the dotted curve but is undistinguishable because these two curves almost perfectly overlap under such scaling. To examine the difference in

detail, an absolute error (AE) curve is shown in Fig.(4.4). It is clear that the reduction of Gibbs phenomenon is remarkable.

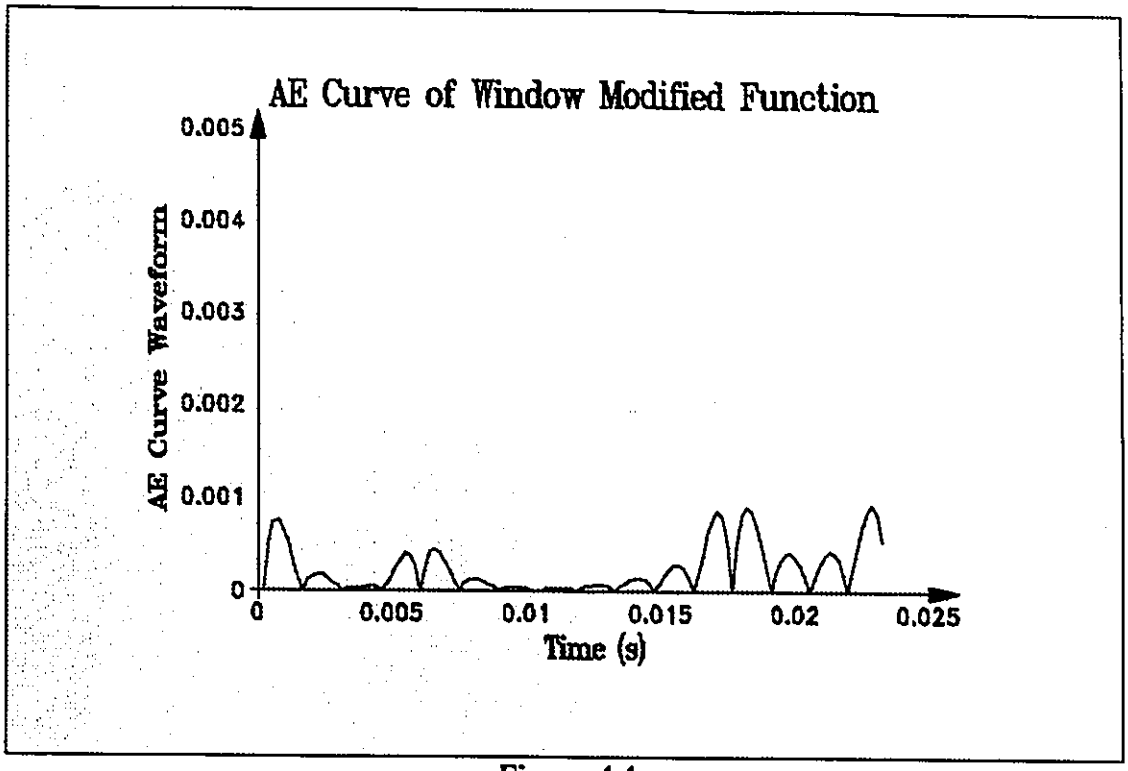

Figure 4.4

To recover the original function, a reverse procedure to windowing is needed, which means

$$f(t) = f_w(t)/w(t) \qquad 0 \le t < T \tag{4.6}$$

However, it is impossible to recover f(0) of the original function from the windowed signal when f(0) was not zero before windowing. In addition, as shown in Fig.(4.5), the reverse windowing to obtain the original function f(t) from $f_w(t)$ introduces recovering errors at the vicinity of two ends of the function.

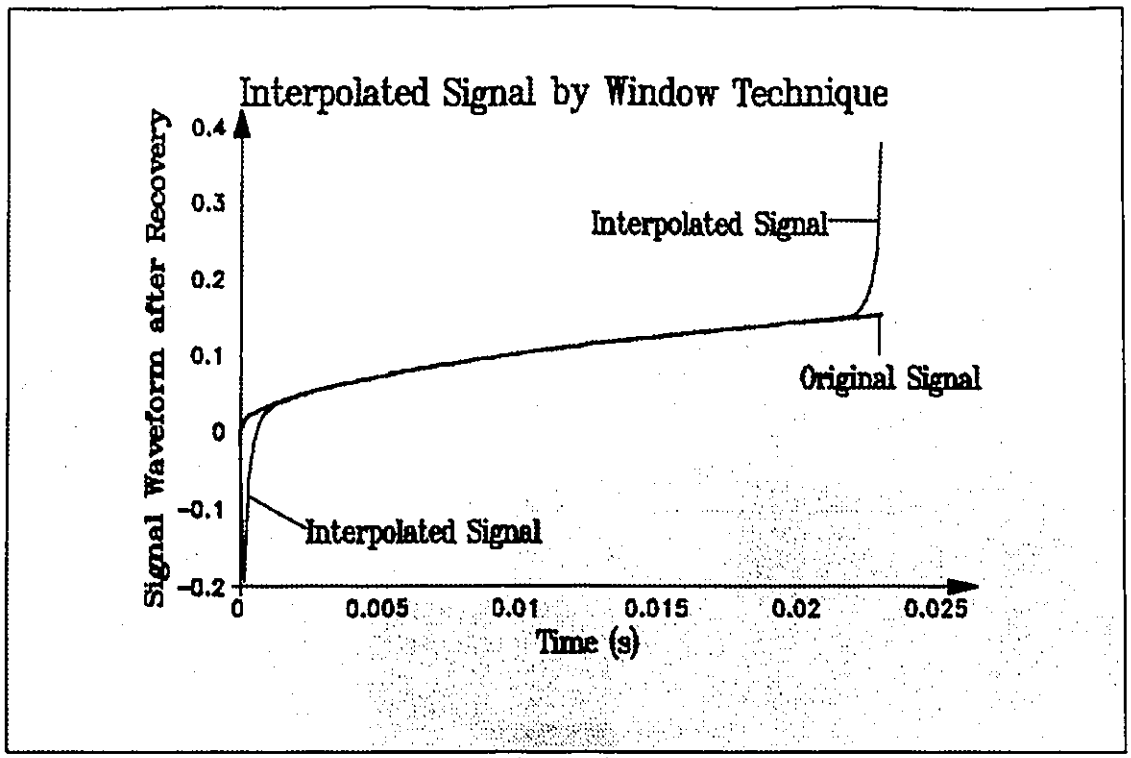

Figure 4.5

4.3 A LINEAR SEQUENCE TRANSFORM

It is already noticed that the jump between f(0) and f(T) is the major reason responsible for the interpolation error. Rather than multiplying a window function on the finite duration function as discussed in the last section, let us consider a linear transform as[31]

$$f'(t) = f(t) + \frac{f(0) - f(T)}{T} t \qquad 0 \le t < T \tag{4.7}$$

It can be noticed that such a transform also makes the transformed function f'(t) possess

the property of f'(0)=f'(T) and hence also eliminates the discontinuity at the periodical boundary if the discrete Fourier transform is applied to f'(t). Fig.(4.6) illustrates the original function and the transformed function.

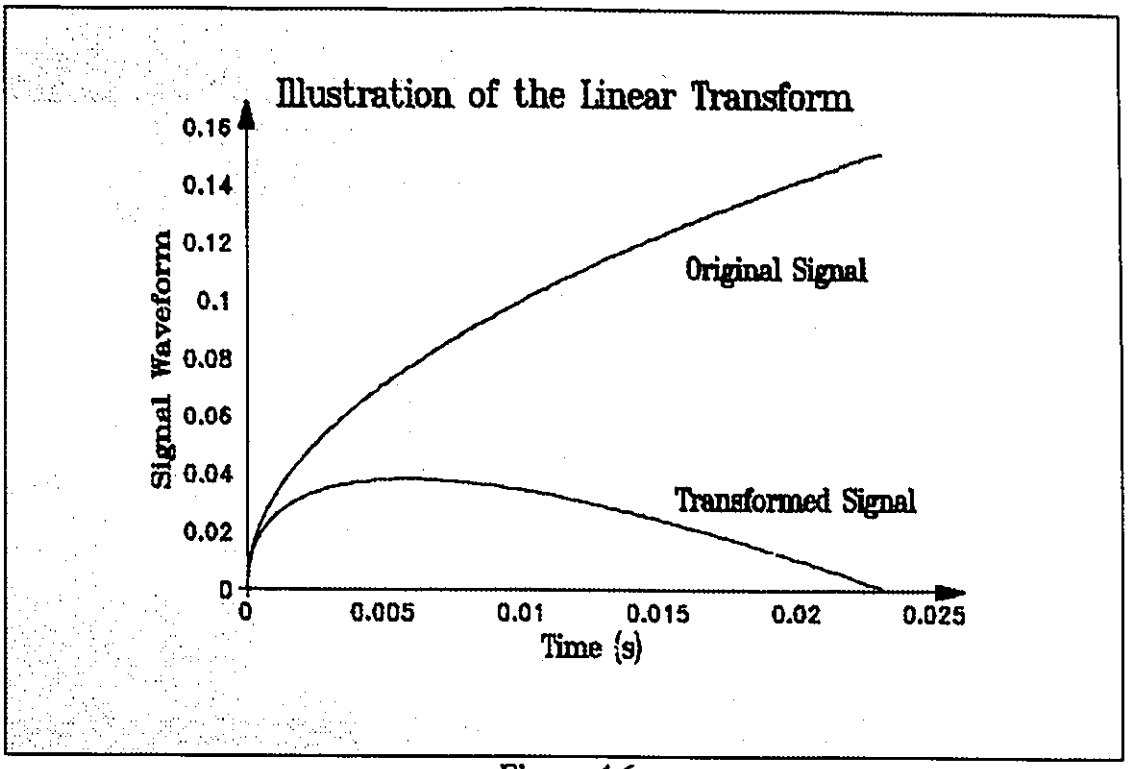

Figure 4.6

Fig.(4.7) shows the AE curve of the reconstruction error for the transformed function obtained from its discrete Fourier transform coefficients. Note that the error is also greatly reduced as compared to the large ripples in Fig.(4.1). In addition, since the transform is linear, the function can be recovered without any error as

$$f(t) = f'(t) - \frac{f'(0) - f'(T)}{T} t \qquad 0 \le t < T \qquad \textbf{(4.8)}$$

where f'(T) is set to f(T) which is not changed in Eq.(4.7). The recovered function overlaps on the original function under same scaling as Fig.(4.5). In the next section, more details will be discussed in terms of experimental study.

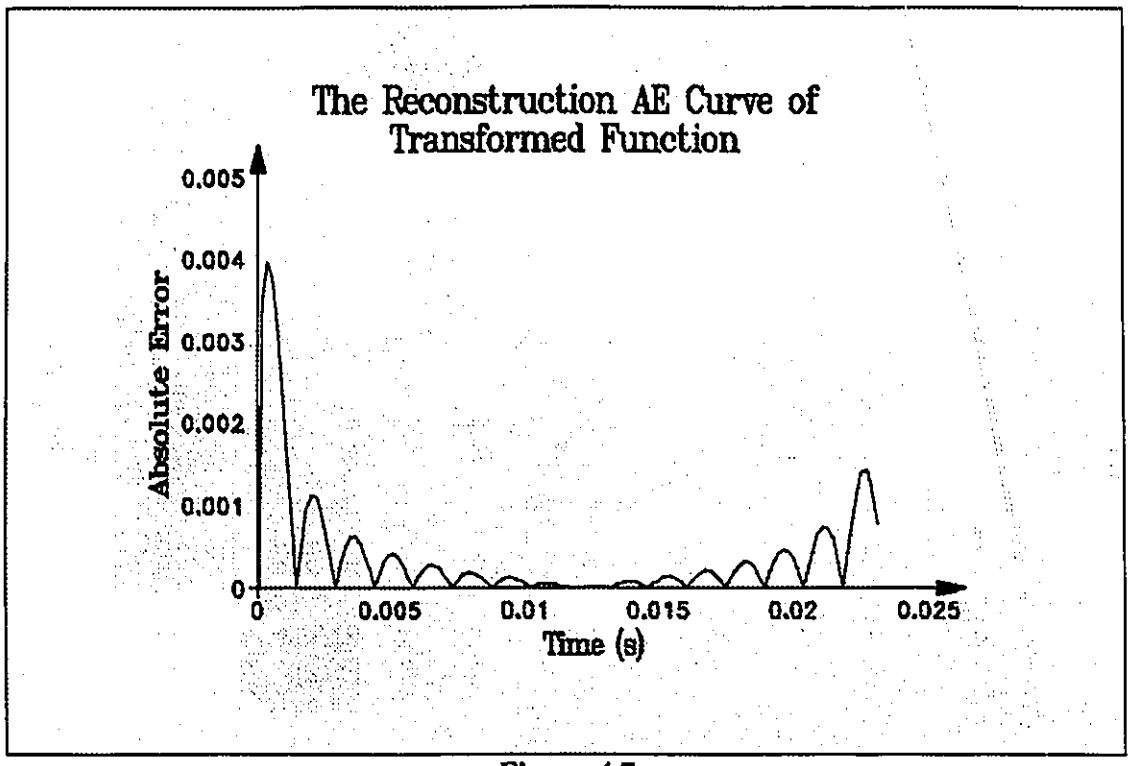

Figure 4.7

4.4 EXPERIMENTAL STUDY

In these experiments, the same cosine test signal is chosen as defined in Eq.(3.23). The same experimental environment is also chosen, which means the sampling rate is f_s=5512.5(Hz), sampling rate increasing factor is P=8, and the number of samples is N_s=128 under sampling rate f_s. The signal frequency varies in the range of 300-3000 (Hz).

Recall from the discussion of last chapter that a different phase shift results in a different interpolation error. The reason is that the signal phase shift alters the initial and the ending values of the signal in the finite duration system and hence changes the height of the jump at the periodic boundary; therefore, interpolation error, the height of ripples at the vicinity of the discontinuity, also changes.

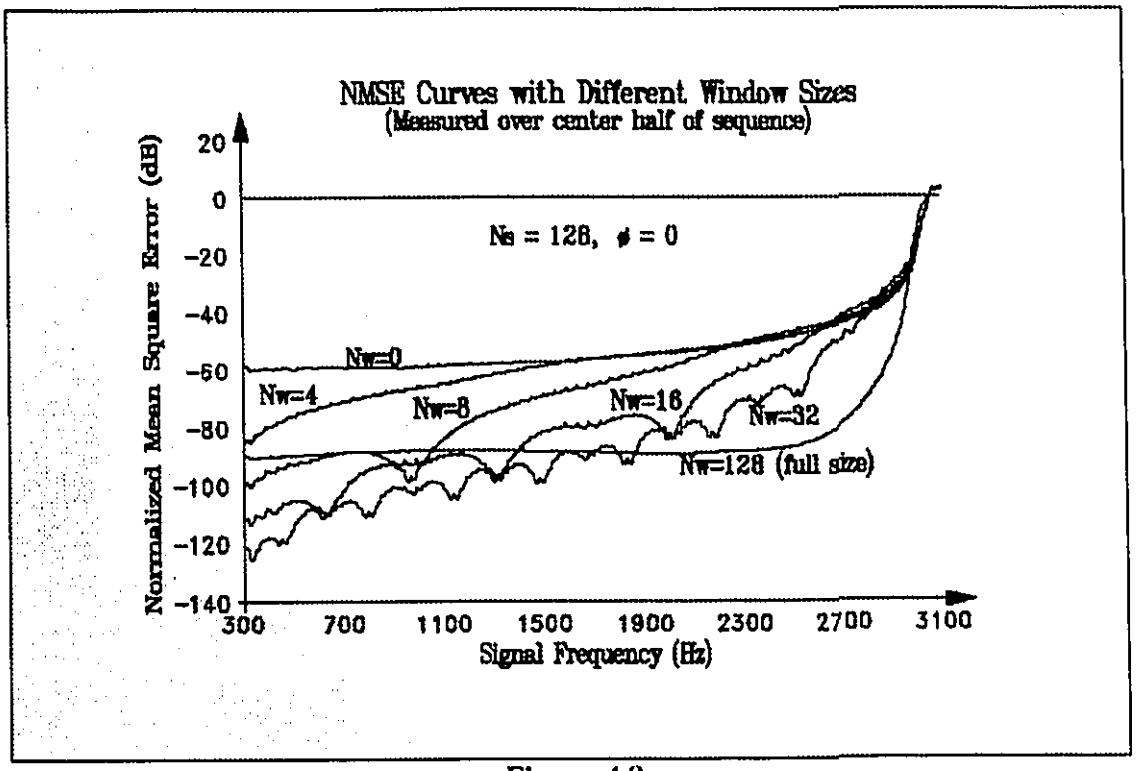

Figure 4.8

Fig.(4.8) shows the smoothed NMSE curves with different window sizes. In general, larger window size results in lower interpolation error. With full window the error curve is reduced throughout the range of the signal frequency and keeps the same

curve trend compared to the usual curve with no windowing. It is encouraging that even a small window (Nw=4) makes significant improvement in accuracy. But larger windowing needs more computation. Note that all curves are measured over centre half of the sequence, which means the large tilt up caused by the recovery procedure (Eq. (4.6)) is mostly neglected in Fig.(4.8).

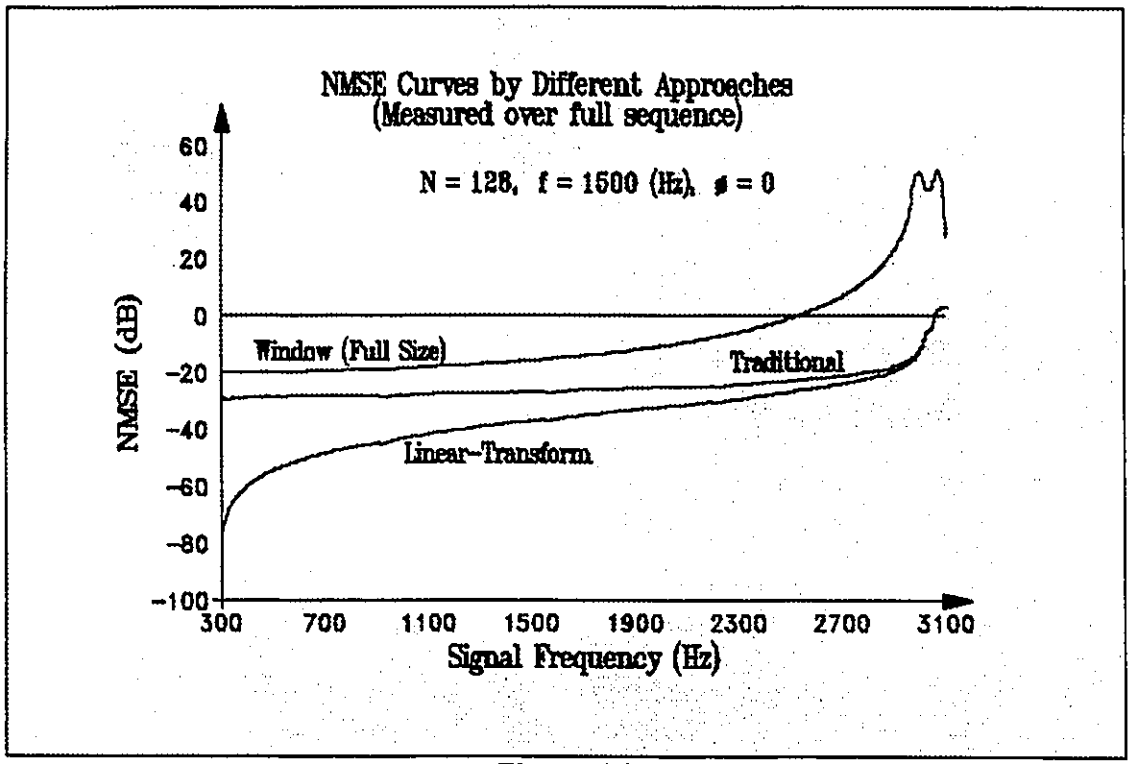

Figure 4.9

Fig.(4.9) and Fig.(4.10) show the NMSE curves measured over the full and the centre half of the sequence respectively by the traditional approach, windowing technique, and the linear sequence transform. It is obvious that the linear sequence transform results

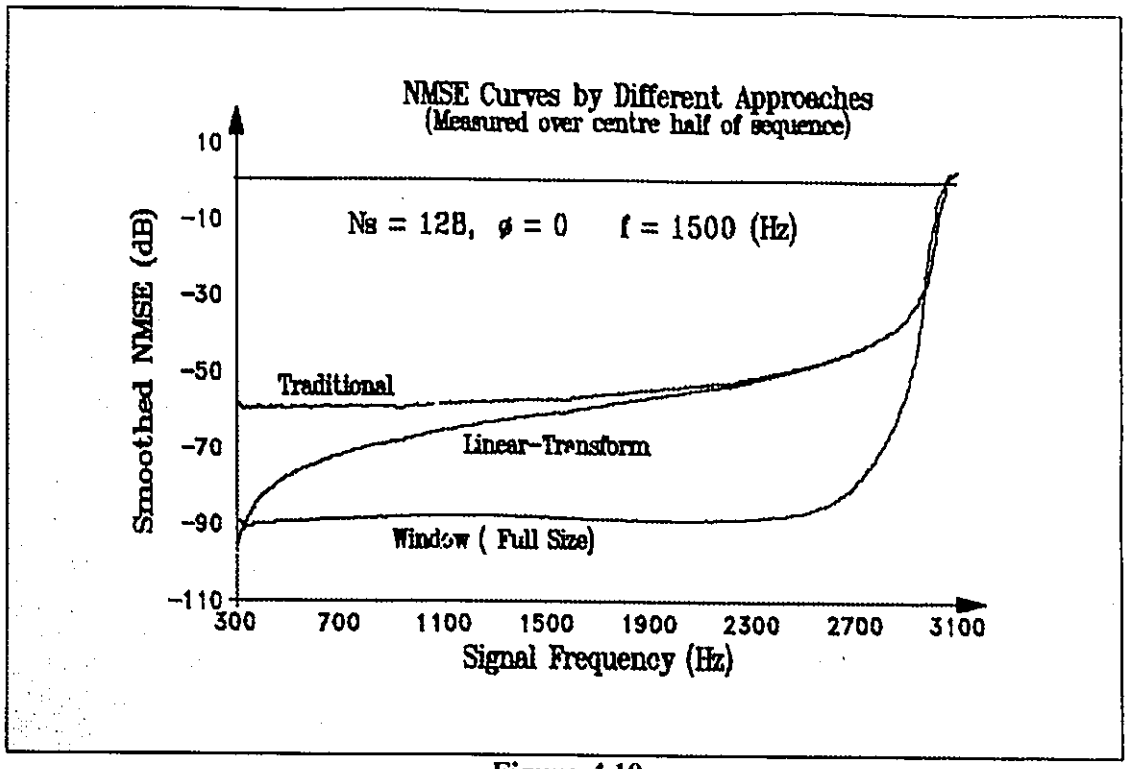

Figure 4.10

in the lowest error overall since the recovery error generated by the reverse windowing seems unacceptable because a large portion of windowing curve is above 0dB line (which means failed interpolation) in Fig.(4.9) although the windowing reduces Gibbs phenomena more than the linear sequence transform, which may be concluded from Fig.(4.10). Besides, from the computation point of view, the windowing obviously needs one multiplication for each sample of the sequence as shown in Eq.(4.4). It seems that the linear sequence transform also needs one multiplication for each sample, however, the multiplication can be implemented in terms of addition by the following programming technique

$$f'(t_i) = f'(t_{i-1}) - \frac{f(T) - f(0)}{T} \tag{4.9}$$

where the initial value f'(0)=f(0). Therefore, the linear transform has another advantage over windowing in terms of computation complexity.

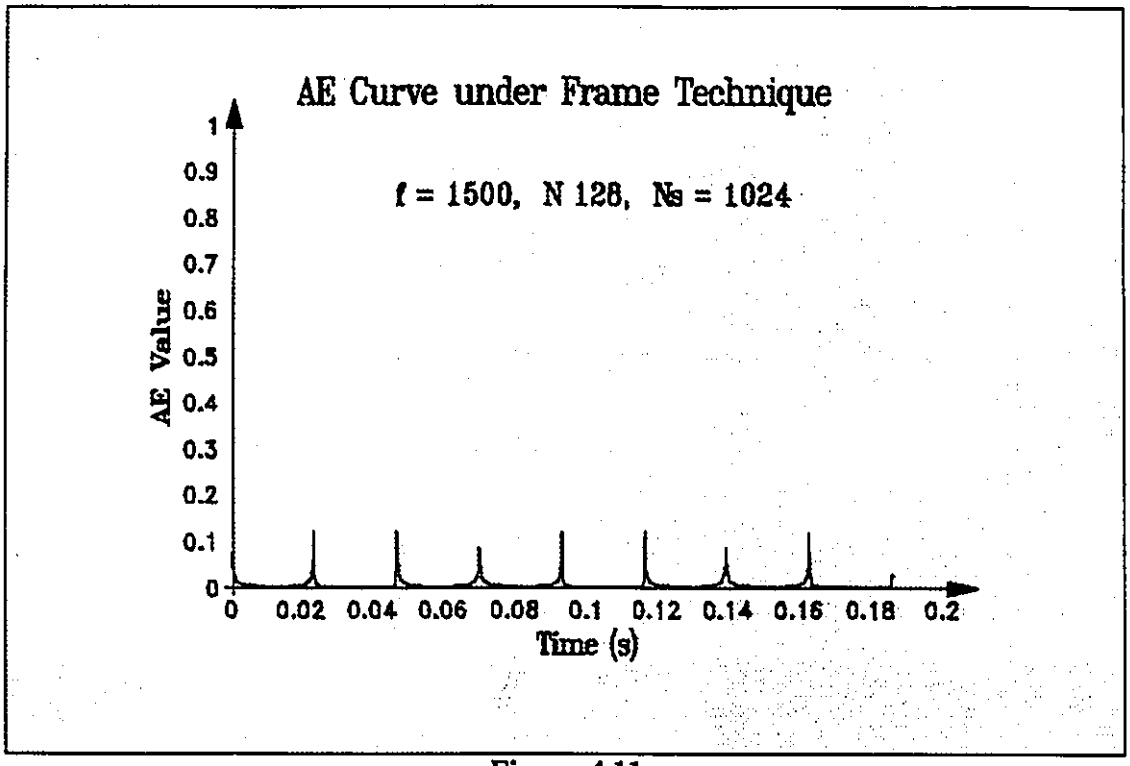

Figure 4.11

As mentioned in the last chapter, longer signals can be divided into shorter frames with reasonable frame length (e.g., N_s=128). Since window technique introduces recovery error, it is not suitable for frame consideration because the peak value at the frame edge could be unacceptable. Only the linear sequence transform technique is considered for using with frames in these experiments. However, the frame consideration under the linear

sequence transform is slightly different from that given in the previous section. Note that f(T) is used in the computation for the transform constant (i.e., (f(T)-f(0))/T) and it is same as f(0) in the next frame. This means that there is one sample overlapped between every two adjacent frames and the function must be defined in the closed interval [0,T] for the transform of last frame, which is different from the half closed interval of [0,T) in the traditional definition of the finite duration function as shown in the last chapter. Fig.(4.11) shows the AE curves over the whole sequence. Note that the peak value at the edges of frame is only 10% of the maximum signal amplitude reduced from 80% as shown in Fig.(3.4).

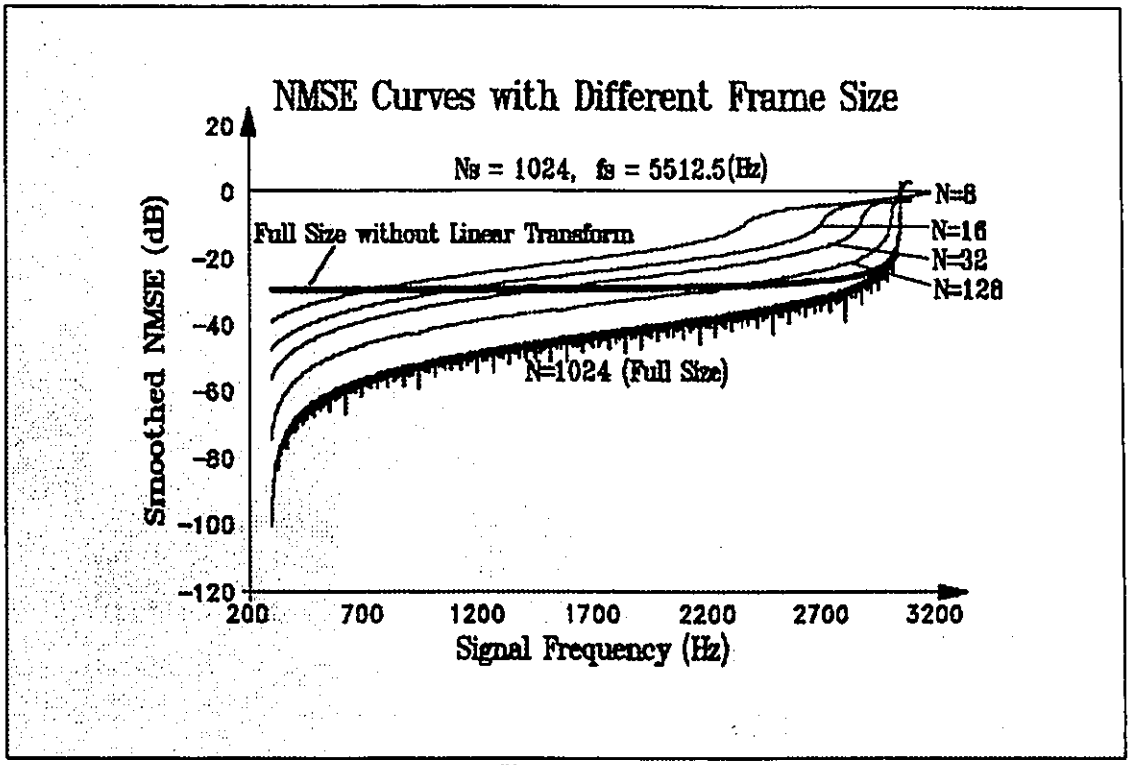

Figure 4.12

Fig.(4.12) shows a group of NMSE curves with different frame sizes using the linear sequence transform technique. Similar to not performing linear transform, smaller frame size results in a larger error when the linear transform is performed. However, it can also be observed that the interpolation error using linear transform is still less than that without using the linear sequence transform when the frame technique is applied. Therefore, it can be concluded that the linear sequence transform not only greatly reduces the interpolation error, but is also suitable for the frame analysis of long signals, while the windowing technique appears not to be suitable.

CHAPTER 5

TWO - DIMENSIONAL INTERPOLATION APPLIED TO IMAGE ZOOMING

5.1 TWO-DIMENSIONAL DFT BASED INTERPOLATION ALGORITHM

From a band-limited continuous 2D signal $f(t_1,t_2)$ of finite duration (T_1,T_2) with discrete version $f(n_1,n_2)$ of length (N_1,N_2), which is obtained by sampling both variables of $f(t_1,t_2)$ with a uniform sampling rate f_s [1], we want to obtain a 2D sequence $g(n_1,n_2)$ of length $(M_1=P\cdot N_1,M_2=P\cdot N_2)$, which can be understood as sampling $f(t_1,f_2)$ at a higher sampling rate f'_s $(f'_s=P\cdot f_s)$. It is obvious that the sequence $f(n_1,n_2)$ only provides one sample for every P^2 samples in sequence $g(n_1,n_2)$, others must be interpolated by the interpolation process from $f(n_1,n_2)$. It should be understood that after interpolation the sequence $g(n_1,n_2)$ must possesses the property of $g(Pn_i,Pn_j)=f(n_i,n_j)$ $\{i=0,1,...,N_1-1,j=0,1,...,N_2-1\}$.

To interpolate the $f(n_1,n_2)$, first compute its 2D DFT as

---

1 We assume that the sampling rate is the same for both variables and above the Nyquist frequency in order to keep the notation simple.

$$F(k_1,k_2)=\frac{1}{N_1N_2}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}f(n_1,n_2)W_{N_1}^{n_1k_1}W_{N_2}^{n_2k_2} \qquad (5.1)$$

$$k_1=0,1,...,N_1-1$$
$$k_2=0,1,...,N_2-1$$

where

$$W_N=e^{-j2\pi/N} \qquad (5.2)$$

Note that since the sequence $f(n_1,n_2)$ is real, the Fourier coefficients $F(k_1,k_2)$ possess the Hermitian symmetry of

$$F(k_1,k_2) = F^*(N_1-k_1,N_2-k_2) \qquad (5.3)$$

$$k_1=0,1,\cdot,\cdot,\cdot,N_1/2$$
$$k_2=0,1,\cdot,\cdot,\cdot,N_2/2$$

Secondly, construct an intermediate coefficient sequence $G(k_1,k_2)$ of length (M_1,M_2) by zero padding $F(k_1,k_2)$. By defining

$$S_i=\{k_i|k_i=0,1,...,N_i/2-1\}$$
$$Q_i=\{k_i|k_i=M_i-N_i/2+1,...,M_i-1\} \qquad (5.4)$$

the zero padded sequence G(k1,k2) is expressed as

$$G(k_1,k_2) = \begin{cases} F(k_1,k_2) & k_1 \in S_1, k_2 \in S_2 \\ F(k_1+N_1-M_1,k_2) & k_1 \in Q_1, k_2 \in S_2 \\ F(k_1,k_2+N_2-M_2) & k_1 \in S_1, k_2 \in Q_2 \\ F(k_1+N_1-M_1,k_2+N_2-M_2) & k_1 \in Q_1, k_2 \in Q_2 \\ 0.5F(k_1,N_2/2) & k_1 \in S_1, k_2 = N_2/2, M_2-N_2/2 \\ 0.5F(N_1/2,k_2) & k_1 = N_1/2, M_1-N_1/2, k_2 \in S_2 \\ 0.5F(k_1+N_1-M_1,N_2/2) & k_1 \in Q_1, k_2 = N_2/2, M_2-N_2/2 \\ 0.5F(N_1/2,k_2+N_2-M_2) & k_1 = N_1/2, M_1-N_1/2, k_2 \in Q_2 \\ 0.25F(N_1/2,N_2/2) & k_1 = N_1/2, M_1-N_1/2 \\ & k_2 = N_2/2, M_2-N_2/2 \\ 0 & others \end{cases} \quad (5.5)$$

It can be noticed from the above equation that the sequence $G(k_1,k_2)$ preserves the Hermitian symmetry from $F(k_1,k_2)$, which means

$$G(k_1,k_2) = G^*(M_1-k_1,M_2-k_2) \quad (5.6)$$
$$k_1 = 0,1,\ldots,M_1/2$$
$$k_2 = 0,1,\ldots,M_2/2$$

Finally, compute the inverse discrete Fourier transform of the intermediate sequence $G(k_1,k_2)$ as

$$g(n_1,n_2)=\sum_{k_1=0}^{M_1-1}\sum_{k_2=0}^{M_2-1} G(k_1,k_2)W_{M_1}^{-n_1k_1}W_{M_2}^{-n_2k_2} \tag{5.7}$$

Therefore we obtain a sequence $g(n_1,n_2)$ of size (M_1,M_2). To prove the obtained sequence $g(n_1,n_2)$ is the desired interpolated sequence, substituting Eq.(5.5) in Eq.(5.6), we have

$$\begin{aligned}
g(Pn_1,Pn_2) &= \sum_{k_1=0}^{M_1-1}\sum_{k_2=0}^{M_2-1} G(k_1,k_2)W_{M_1}^{-Pn_1k_1}W_{M_2}^{-Pn_2k_2} \\
&= \sum_{k_1=0}^{M_1-1}\sum_{k_2=0}^{M_2-1} G(k_1,k_2)W_{N_1}^{-n_1k_1}W_{N_2}^{-n_2k_2} \\
&= \sum_{\substack{k_1\in S_1,Q_1\\ k_2\in S_2,Q_2}} G(k_1,k_2)W_{N_1}^{-n_1k_1}W_{N_2}^{-n_2k_2} \\
&\quad + (-1)^{-n_2}\sum_{k_1\in S_1\cup Q_1}\left[G(k_1,N_2/2) + G(k_1,M_2-N_2/2)\right]W_{N_1}^{-n_1k_1} \\
&\quad + (-1)^{-n_1}\sum_{k_2\in S_2\cup Q_2}\left[G(N_1/2,k_2) + G(M_1-N_1/2,k_2)\right]W_{N_2}^{-n_2k_2} \\
&\quad + (-1)^{n_1+n_2}\left[\begin{array}{c} G(N_1/2,N_2/2) + G(M_1-N_1/2,N_2/2) + \\ G(N_1/2,M_2-N_2/2) + G(M_1-N_1/2,M_2-N_2/2)\end{array}\right]
\end{aligned} \tag{5.8}$$

By defining another group as

$$P_i \in \{k_i:\ k_i = N_i/2+1, N_i/2+2, \cdots, N_i-1\} \tag{5.9}$$

the Eq.(5.8) can be expressed as

$$\begin{aligned} g(Pn_1,Pn_2) &= \sum_{k_1\in S_1,P_1}\sum_{k_2\in S_2,P_2} F(k_1,k_2)W_{N_1}^{-n_1k_1}W_{N_2}^{-n_2k_2} \\ &\quad +(-1)^{n_1}\left[\sum_{k_2\in S_2,P_2} F(N_1/2,k_2)W_{N_2}^{-n_2k_2}\right] \\ &\quad +(-1)^{n_2}\left[\sum_{k_1\in S_1,P_1} F(k_1,N_2/2)W_{N_1}^{-n_1k_1}\right] \\ &\quad +F(N_1/2,N_2/2) \\ &= \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} F(k_1,k_2)W_{N_1}^{-n_1k_1}W_{N_2}^{-n_2k_2} \\ &= f(n_1,n_2) \end{aligned} \tag{5.10}$$

Therefore, the obtained sequence $g(n_1,n_2)$ is the desired interpolated sequence from $f(n_1,n_2)$.

5.2 TWO-DIMENSIONAL LINEAR SEQUENCE TRANSFORM

As discussed in the one-dimensional case, the discontinuity at the periodical boundary results in the large oscillation error for the inverse discrete Fourier transform

approximation. It has been also shown before that the elimination of the discontinuity achieves much higher interpolation accuracy. Therefore, the reduction of the discontinuity at the periodic boundary for two-dimensional function should also be considered. However, the periodic expansion of a two-dimensional signal is different from the one-dimensional case, because there exist different boundary samples. Hence different considerations must be taken.

Let $f(t_1,t_2)$ be a finite duration two-dimensional function with $0 \le t_1 < T_1$ and $0 \le t_2 < T_2$. When the discrete Fourier transform is applied to this function, the function is assumed to be in the one period of the periodical function as

$$f_p(t_1,t_2) = f(t_1,t_2) \otimes \delta_{T_1 T_2}(t_1,t_2) \qquad \textbf{(5.11)}$$

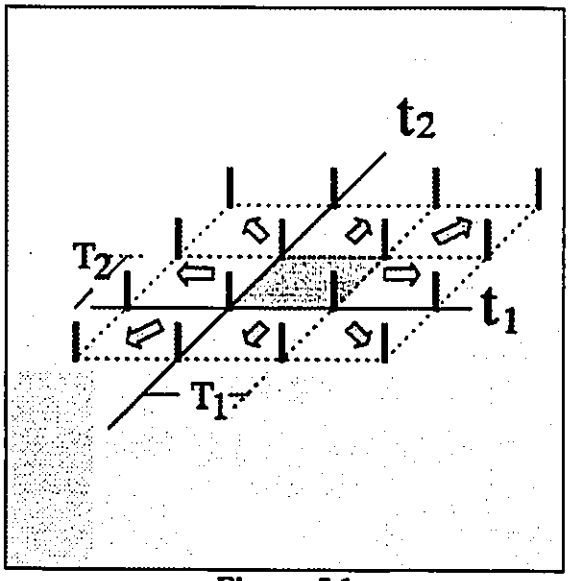

Figure 5.1
Periodic expanding of the two-dimensional function

where $T(t_1,t_2)$ is illustrated in Fig.(5.1). Note that there are several possible discontinuity jumps in the periodic function $f_p(t_1,t_2)$. These discontinuities could be the differences between $f(0,t_2)$ and $f(T_1,t_2)$, $f(t_1,0)$ and $f(t_1,T_2)$, $f(0,0)$ and $f(T_1,T_2)$ and $f(0,T_2)$ and $f(T_1,0)$. To eliminate the discontinuities, it must be ensured that $f(0,t_2)=f(T_1,t_2)$, $f(t_1,0)=f(t_1,T_2)$ and $f(0,0)=f(0,T_2)=f(T_1,0)=f(T_1,T_2)$.

If the window function described in Eq.(4.5) is applied to each row and column for the sampled two-dimensional function separately, all boundary values are set to zero. The discontinuities are eliminated and hence the interpolation for the windowed function is reduced. But windowing also introduces the recovery error as discussed in one-dimensional case. It will be shown in the experimental results later in this chapter that the recovery error results in an unacceptable effect on the image.

Now consider a two-dimensional linear transform[32,33]

$$f'(t_1,t_2) = f(t_1,t_2)+c_1(t_1)\cdot t_2+c_2(t_2)\cdot t_1+c_0\cdot t_1\cdot t_2 \tag{5.12}$$

where the transform coefficients are obtained by

$$\begin{aligned} c_1(t_1) &= \frac{f(t_1,0)-f(t_1,T_2)}{T_2} && 0\le t_1<T_1 \\ c_2(t_2) &= \frac{f(0,t_2)-f(T_1,t_2)}{T_1} && 0\le t_2<T_2 \\ c_0 &= \frac{f(0,0)-f(0,T_2)-f(T-1,0)+f(T_1,T_2)}{T_1\cdot T_2} \end{aligned} \tag{5.13}$$

It is easy to see that the new function $f'(t_1,t_2)$ possesses the property of

$$\begin{aligned} f'(0,t_2) &= f'(T_1,t_2) && 0\le t_2<T_2 \\ f'(t_1,0) &= f'(t_1,T_2) && 0\le t_1<T_1 \end{aligned} \tag{5.14}$$

In other words, the function $f'(t_1,t_2)$ is continuous at the periodical edges when it is

extended periodically and thus the Gibbs phenomenon is eliminated when the discrete Fourier transform is performed. Therefore the error of the interpolated two-dimensional signal will be reduced as it has already been observed in the one-dimensional case.

To recover the original function $g(t_1,t_2)$ from the transformed-interpolated function $g'(t_1,t_2)$, similar to the one-dimensional case, a reverse linear transform can be done as

$$g(t_1,t_2) = g'(t_1,t_2) - d_1(t_1)\cdot t_2 - d_2(t_2)\cdot t_1 - d_0\cdot t_1\cdot t_2 \qquad \begin{matrix} 0\le t_1<T_1 \\ 0\le t_2<T_2 \end{matrix} \qquad \textbf{(5.15)}$$

where

$$d_1(t_1) = \frac{g'(t_1,0)-g'(t_1,T_2)}{T_2} \qquad 0\le t_1<T_1$$

$$d_2(t_2) = \frac{g'(0,t_2)-g'(T_1,t_2)}{T_1} \qquad 0\le t_2<T_2 \qquad \textbf{(5.16)}$$

$$d_0 = \frac{g(0,0)-g(0,T_2)-g(T_1,0)+g(T_1,T_2)}{T_1\cdot T_2}$$

The value of $g'(T_1,T_2)$ is set to $f(T_1,T_1)$ as $g'(T)$ is set to $f(T)$ in the one-dimensional case. However, $g'(t_1,T_2)$ and $g'(T_1,t_2)$ cannot simply set to $f(t_1,T_2)$ and $f(T_1,t_2)$ respectively because they are also time functions, although only for one variable, and the function is under new sampling rate. Therefore, one-dimensional interpolation on $f(t_1,T_2)$ and $f(T_1,t_2)$ must be performed in order to obtain the linear transform coefficients $d_1(t_1)$

and $d_2(t_2)$. Since there exits interpolation error on these two coefficients, although it can be reduced by employing one-dimensional linear transform technique, the recovery of a two- dimensional function also introduces error to the entire interpolation process. But as we will see in the next section, the error introduced by the recovery is not as much as that introduced by the windowing, and usually is acceptable.

5.3 IMAGE ZOOMING BY TWO-DIMENSIONAL INTERPOLATION

Image zooming is a part of practical image processing on geometric processes or image scaling[34]. Since the interpolation of a two-dimensional function is to approximate the sample value under higher sampling rate from an already sampled function, it is suitable for the purpose of image zooming, which is to insert pixels between the existing pixels. In this section, the two-dimensional discrete Fourier transform based interpolation is applied to image zooming.

The test image is a subimage cut from the common human face "lena" image as marked in the bright frame in Fig.(5.2). The size of the test image is N1*N2 (N1=N2=64 pixel in our experiments). The zooming ratio P is fixed to 8 in our experiments, which means the zoomed image is the size of 256*256. Since the image under the desired higher sampling rate is not available, the visual effect becomes the judgement of the zooming quality, which is also common in image processing.

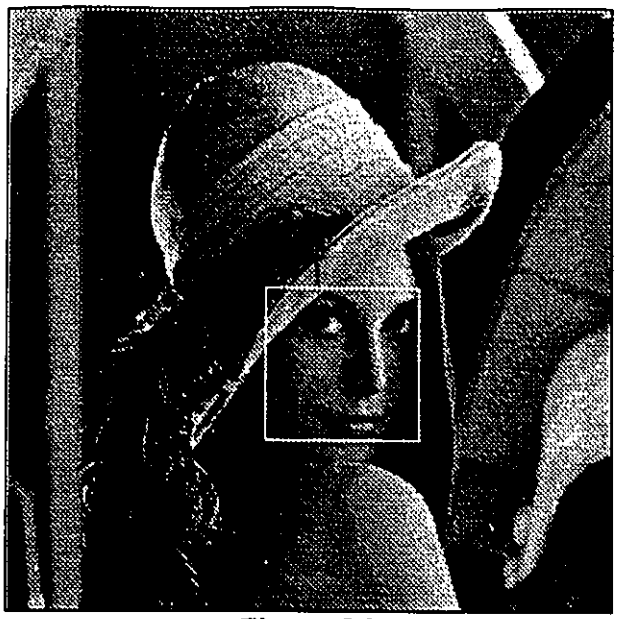

Figure 5.2
The Original Lena Image

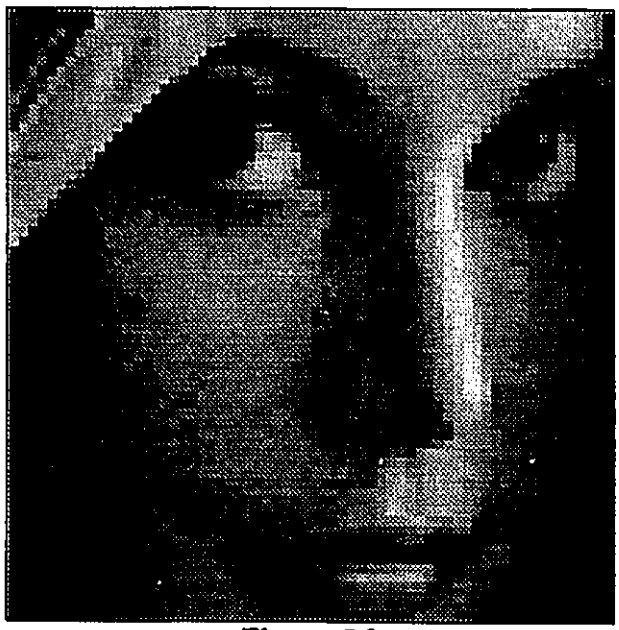

Figure 5.3
Zoomed image by pixel duplication

In image processing, the most common method for image zooming is called pixel duplication. That is, to repeat each pixel on both directions several times. This indeed saves computation time but usually does not achieve good results. Fig.(5.3) shows the zoomed image using pixel duplication. It can be noticed that details are blurred under this method and zooming zig-zaging can be observed at the edges of the image.

Figure 5.4
Zoomed Image by DFT Interpolation

Fig.(5.4) shows the image zoomed by two-dimensional interpolation using the conventional DFT method. Much better results are achieved than the pixel duplication method. However, as mentioned before, the Gibbs phenomenon, i.e., the interpolation error caused by discontinuity at the periodical boundary, results in the ripples at the edge of the image.

Fig.(5.5) shows the image zoomed from DFT based interpolation using windowing technique. The size of window is half the size of the image. Because of the recovery error from the reverse windowing process, i.e., the big tilt up at the ends of the sequence in one-dimensional case, a vignetting effect is observed in the image. It is known that the larger the window size, the lower the interpolation error for the one-dimensional function. However, for two-dimensional images, larger size windows give a worse visual effect. This can be observed from the image zoomed under full window size shown in Fig.(5.5). The reasons are that first, the smaller window (e.g., size of 4) limits the tilt up in the smaller area (only within two rows or columns at each edge of the image for the size of 4 window); second, the errors at the centre of the image are below the quantization level.

Figure 5.5
Zoomed image under windowing
Window size = 64 (Full window)

Figure 5.6
Zoomed image under windowing
Window size = 4

Fig.(5.7) shows the zoomed image by the linear transform method. To use the linear transform technique, the size of the test image must be $(N_1+1)*(N_2+1)$ (the extra row and column represents $f(t_1,T_2)$ and $f(T_1,t_2)$ respectively in Eq.(5.12)). The extra row and column is obtained by cutting the test image from standard image at the desired size in our experiments. In the situation where there is no larger image available to cut from, (e.g. to zoom the whole existing image), the extra row and column can be obtained by extrapolation. Although there are errors in the recovery process caused by the interpolation error in the last row and the last column under the linear transform technique, the errors seem also below the quantization level. Therefore, a superior smooth image is observed: there are no visible ripples and there is no vignetting effect.

Figure 5.7
Zoomed image by linear transform

Block technique is a widely used technique in image processing because of the computation and memory saving by dividing large size image into smaller size blocks. The block technique, which is similar to the frame technique in one dimensional signal interpolation, can also be applied to image zooming .

Fig.(5.8) shows the zoomed image using the block technique by the traditional two- dimensional interpolation scheme. The image is first divided into blocks with a size of Nb_1*Nb_2, where $N_1/Nb_1=K_1$ and $N_2/Nb_2=K_2$, then each block is interpolated separately to obtain the zoomed image. It is obvious that the interpolation error, especially the Gibbs phenomenon at the edge of the block, causes the block effect that severely deteriorates the image.

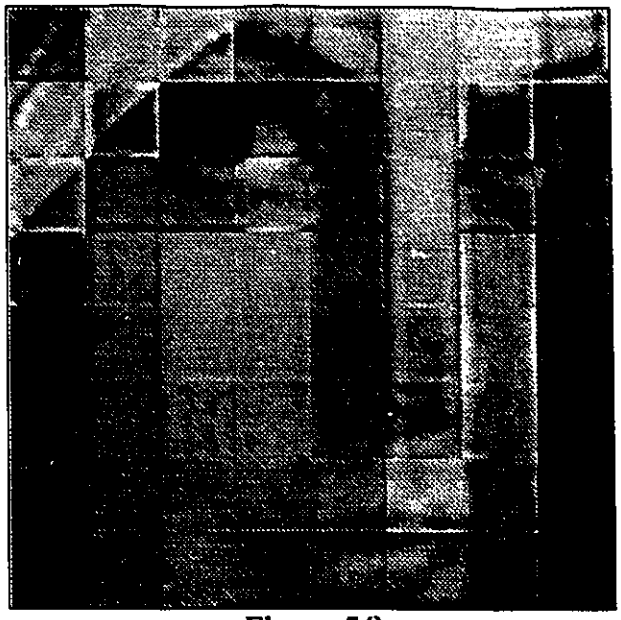

Figure 5.8
Zoomed image by block technique
Block size = 8*8

Figure 5.9
Zoomed image by blocks under
linear transform technique.
Block size = 8*8

To eliminate the block effect, some techniques should be used to reduce the interpolation error, especially at the edge of blocks. Since the vignetting effect at the block edges is caused by the window technique, the windowing is not considered for the block technique. Fortunately, the linear transform technique gives the perfect solution for eliminating the block effect. However, different block divisions and operations must be considered under the linear transform technique because one more row and column are needed for each block to perform the linear transform.

Different from the block size of Nb_1*Nb_2 in the conventional block technique, the image is divided into blocks of size $(Nb_1+1)*(Nb_2+1)$ with one row and column overlap to the previous one. Then each block is transformed and interpolated without considering the last row and last column in a certain order. (The last row and column only participate in the transform and recovery and are interpolated separately as discussed under the two-dimensional linear transform technique.) The zoomed blocks are placed back without overlapping and therefore an intermediate image of size N_1*N_2 is obtained. To recover the desired zoomed image from linear transform, the last row and column of the image must be interpolated by a one-dimensional interpolation (certainly one dimensional linear transform and recovery should be applied) and must be considered as an extra row and column of the intermediate image respectively. Then the two-dimensional recovery transform is performed on each block but in reverse order from transform and zooming order.

Fig.(5.9) shows the zoomed image by using the block technique described above. The image appears smooth and no block effect can be observed.

CHAPTER 6

CONCLUSIONS

6.1 CONCLUSIONS

In this thesis, the fundamental theory of discrete Fourier transform based interpolation is studied in detail. It is shown in this thesis that the discrete Fourier transform based interpolation is the problem of approximating a finite duration function by Fourier series. The discrete Fourier transform coefficients, in fact, are the aliased Fourier series coefficients of a sampled function. By examining the Gibbs phenomenon in detail, which occurs at the vicinity of the discontinuity of the original function, an efficient linear sequence transform is proposed to efficiently suppress the Gibbs phenomenon. Compared to the windowing technique, the proposed linear sequence transform has several advantages. First, the linear sequence transform does not introduce recovery error; second, by applying the programming technique, linear sequence transform achieves higher computation saving; third, the linear sequence transform is suitable for frame operation on a long signal which may imply that signal interpolator can be implemented at a small cost without compromising the accuracy too much.

The interpolation of two-dimensional functions is also studied in this thesis and is applied to the practical problem of image zooming. In correspondence with one-

dimensional linear sequence transform, a two-dimensional linear sequence transform is also proposed. It gives better results than the window method. Besides, it gives a very good solution for eliminating the block effect when using block technique in image zooming.

FUTURE DIRECTION

This work has already provoked some interest in the interpolation problem. Macleod in Cambridge University proposed a further improvement on DFT based interpolation by removing the wraparound discontinuity also in the first derivative at the periodic boundary of assumed periodically extended function[35]. He also implied that such an effect could be extended to remove higher order wraparound discontinuities in derivatives, certainly, with the compromise of the computation involvement. The problem is that there exit errors when obtaining the derivative at a certain point of the function from the discrete values and to calculate high order derivatives, such errors can be accumulated. This implies that removing higher order wraparound discontinuities may not generate better results.

REFERENCES

[1] J.F. Steffensen, *"Interpolation"*, Chelsea Publishing Company, New York, 1950.

[2] P.J. Davis, *"Interpolation and Approximation"*, Blaisdell Publishing Company, New York, ch. 2, pp. 24-53, 1963.

[3] P.J. Davis, *"Interpolation and Approximation"*, Blaisdell Publishing Company, New York, ch. 4, pp. 78-94, 1963.

[4] K. Singhal and J. Vlach, *"Interpolation Using the Fast Fourier Transform"*, Proc. IEEE, December.1972.

[5] J.W. Cooley and J.W. Tukey, *"An Algorithm for the Machine Calculation of Complex Fourier Series"*, Math. Comput. vol. 1, pp. 297-301, 1965.

[6] R.W. Schafer and L.R. Rabiner, *"A Digital Signal Processing Approach to Interpolation"*, Proc. IEEE, Vol. 61, No. 6, June 1973.

[7] K.P. Prasad and P. Satyanarayana, *"Fast Interpolation Algorithm using FFT"*, Electronics Letters, Vol. 22, No. ?, pp.185-187, 1986.

[8] D. Fraser, *"Interpolation by the FFT Revisited - An Experimental Investigation"*, IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-37, No. 5, pp. 665-675, May 1989.

[9] P. Sathyanarayana, P.S. Reddy and M.N.S. Swamy, *"Interpolation of 2-D Signals"*, IEEE Trans. Circuits and Systems, Vol. 37, No. 5, pp. 623-625, May 1990.

[10] T. Smit, M.R. Smith and S.T. Nichols, *"Efficient Sinc Function Interpolation Technique for Centre Padded Data"*, IEEE. Trans. Acoust. Speech, Signal

Processing, Vol. ASSP-38, No. 9, pp. 1512-1517, September 1990.

[11] D.P. Skinner, *"Pruning the decimation-in-time FFT algorithm"*, IEEE Trans. Acoust. Speech, Signal Processing, Vol. ASSP-24, pp.193-194, April 1976.

[12] S.C. Chan, K.L. Ho and C.W. Kok, *"Interpolation of 2-D Signal by Subsequence FFT"*, IEEE Trans. Circuits and Systems-II, Vol. 40, No. 2, pp.115-118, Feb. 1993.

[13] A.E. Rosenberg, R.W. Schafer and L.R. Rabiner, *"Effects of Smoothing and Quantizing the Parameters of Formant-Coded Voiced Speech"*, J. Acoust. Soc. Amer., Vol. 50, pp. 1532-1538, Dec. 1971.

[14] R.W. Schafer and L.R. Rabiner, *"Design and Simulation of a Speech Analysis -Synthesis System Based on Short-time Fourier Analysis"*, IEEE Trans. Audio Electroacoust., Vol. AU-21, pp. 165-174, June 1973.

[15] S.L. Freeney, et. al., *"Design of Digital Filters for an All Digital Frequency Division Multiplex-time Division Multiplex Translator"*, IEEE Trans. Circuit Theory, Vol. CT-18, pp. 702-711, Nov. 1971.

[16] S.L. Freeney, J.F. Kaiser and H.S. McDonnald, *"Some Applications of Digital Signal Processing"*, A.V. Oppenheim, Ed. Englewood Cliffs, NJ: Prentice-Hall, ch. 1, pp. 1-28, 1978.

[17] R.G. Pridham and R.A. Mucci, *"Digital Interpolation Beamforming for Lowpass and Bandpass Signals"*, Proc. IEEE, Vol. 67, No. 6, pp. 904-919, June 1979.

[18] L.R. Rabiner and B. Gold, *"Theory and Application of Digital Signal Processing"*, Englewood Cliffs, NJ: Prentice-Hall, ch. 13, 1975.

[19] R.W. Hamming, *"Numerical Methods for Scientists and Engineers"*, McGraw-Hill, New York, Ch. 3, pp.503-561, 1973.

[20] A. Papoulis, *"Signal Analysis"*, McGrae-Hill, New York, ch. 3, pp. 56-100, 1977.

[21] J. Arsac, *"Fourier Transforms and the Theory of Distributions"*, Englewood Cliffs, NJ: Prentice Hall, 1966.

[22] E.O. Brigham, *"The Fast Fourier Transform"*, Englewood Cliffs, NJ: Prentice Hall, ch. 10, 1974.

[23] H. Freeman, *"Discrete-Time Systems"*, Wiley, New York, 1965.

[24] F.J. Harris, *"Time Domain Signal Processing with the DFT"* in *"Handbook of Digital Signal Processing - Engineering Applications"*, Ed. D.F. Elliot, Academic Press, New York, ch. 8, pp.663-698, 1987.

[25] J.W. Gibbs, " - ", Nature, Vol. 59, p. 606, 1899.

[26] H. Wilbraham, *"On a Certain Periodic Function"*, Cambridge and Dublin Math. J., Vol. 3, p. 198, 1848.

[27] H.S. Carslaw, *"A Historical Note on Gibbs' Phenomenon in Fourier's Series and Integrals"*, Bull. Amer. Math. Soc., Vol. 31, pp. 420-424, 1925.

[28] Bôcher, *"Introduction to the Theory of Fourier's Series"*, Ann. of Math., Vol. 7, No. 2, p. 81, 1906.

[29] H.S. Carslaw, *"Theory of Fourier's Series and Integrals"*, Dover Publications Inc., ch. 9, pp. 289-308, 1930.

[30] H. Dym and H.P. McKean, *"Fourier Series and Integrals"*, Academic Press, New York, ch. 1, pp. 43-46, 1972.

[31] Z.W. Wang, J.J. Soltis and W.C. Miller, *"Improved Approach to Interpolation using the FFT"*, Elect. Lett., Vol. 28, No. 25, pp. 2320-2322, 1992.

[32] Z. Wang, J.J. Soltis and W.C. Miller, *"Image Zooming by Improved Interpolation Scheme using FFT"*, The 36th Midwest Symposium on Circuits and Systems, Detroit, August 1993.

[33] Z. Wang, J.J. Soltis and W.C. Miller, *"An Improved 2D Interpolation Scheme using FFT"*, Canadian Conf. on Elec. Comp. Engineering, pp. 171-174, Vancouver, September 1993.

[34] C.A. Lindley, *"Practical Image Processing in C"*, John Wiley & Sons, New York, pp. 407-408, 1991.

[35] M.D. Macleod, *"Fast Interpolation by FFT with Greatly Increased Accuracy"*, Electronics Letters, Vol. 29, No. 13, pp.1200-1201, June 1993.

[36] S.T. Tretter, *"Introduction to Discrete-Time Signal Processing"*, John Wiley & Sons, New York, ch. 3, pp. 23-38, 1976.

APPENDIX

C PROGRAM FOR IMAGE ZOOMING

```
/*
 *      C Code for Image Zooming
 *
 *      File name       : Zooming.c
 *      Input File      : A recognizable image file
 *      Output File     : Zoomed image file in same format
 */

#define pi 3.14159265358979    /*  define π   */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void zoom();                   /*  Major function for zooming images */
void extrapolate();            /*  Function for extralopating extra row and column   */
void interpolation();          /*  Function for 2-Dimensional DFT interpolationn     */
void L_transform();            /*  Function for 2-Dimensional linear transform    */
void spectrum();               /*  Spectrum domain zero padding  */
void getprime();               /*  Function compute the prime factor of image size   */
void FFT();                    /*  Function for FFT algorithm  */
void One_D_interpolate();      /*  Function for 1-Dimensional interpolation    */
void output_image();

void main ( argc, argv )

int     argc;
char    *argv[];

{       /* main         */
        FILE   *fp_in, *fp_out;
        char    ch;
        int     width_in, length_in;
        int     depth;
        int     width_out,length_out;
        int     ratio;
        int     i, j;
        unsigned char          head[3];
        unsigned char          *img_in, *img_out;

        if (argc < 2 )
                {
```

```
        printf ( " You forgot to type input and output file name\n" );
        printf ( " on the command line. Try again.\n" );
        exit (1);
        }
if ( argc < 3 )
        {
        printf ( " You forgot to give output file name\n");
        exit (2);
        }
if ((fp_in=fopen(argv[1],"rb+"))==NULL)
        {
        printf ( "cannot pen input file\n" );
        exit (3);
        }
if ((fp_out=fopen(argv[2],"wb"))==NULL)
        {
        printf("cannot open output file\n");
        exit (4);
        }

fgets (head, 3, fp_in);
ch = fgetc(fp_in);
fscanf(fp_in, "%d", &width_in);          /* get image width    */
ch = fgetc(fp_in);
fscanf (fp_in, "%d", &length_in);    /* get image length   */
ch = fgetc(fp_in);
fscanf (fp_in, "%d", &depth);        /* get image depth    */
ch = fgetc(fp_in);

printf(" The current image is %d x %d\n", width_in, length_in);
printf(" Give the zooming ratio (power of 2):");
scanf("%d", &ratio);
printf("\n");

width_out = ratio*width_in;
length_out = ratio*length_in;

img_out = (unsigned char *)malloc(length_out*width_out*sizeof(unsigned char));
if ( !img_out )
        {
        printf("memory request failed (img_out)\n");
```

```
            exit (6);
            }
        img_in = (unsigned char *)malloc((width_in+1)*(length_in+1)*
                sizeof(unsigned char));
        if ( !img_in )
            {
            printf("memory request failed (img_in)\n");
            exit (7);
            }

        for ( i=0; i<length_in; i++)
        for ( j=0; j<width_in; j++ )
            {
            img_in[i*width_in+j] = fgetc ( fp_in );
            }

        zoom(img_in, img_out, width_in, length_in, ratio);

        fputs(head, fp_out);
        fputc(ch, fp_out);
        fprintf(fp_out, "%d", width_out);
        fputc(ch, fp_out);
        fprintf(fp_out, "%d", length_out);
        fputc(ch, fp_out);
        fprintf(fp_out, "%d", depth);
        fputc(ch, fp_out);
        fwrite(img_out, length_out*sizeof(unsigned char), width_out, fp_out);

        fclose(fp_in);
        fclose(fp_out);
        free(img_in);
        free(img_out);

        exit(0);

}       /*      main    */

void zoom(img_in, img_out, width_in, length_in, ratio)   /*  calling from main    */

unsigned char           *img_in, *img_out;
```

```
int     width_in, length_in, ratio;

{       /*      zoom    */
        float   *blk_in, *blk_out;
        int     blk_width_in, blk_length_in;
        int     blk_width_out, blk_length_out;
        int     i,j;
        int     fblk_width_in, fblk_length_in;
        int     fblk_width_out, fblk_length_out;
       int      width_out;


        printf(" The current image size is : %d x %d\n", width_in, length_in );
        printf(" Give the width of block (2 < size < %d )\n", width_in );
        printf(" Block width= ");
        scanf(" %d", &blk_width_in );
        printf(" \n ");
        printf(" Give the length of block (2 < size < %d )\n", length_in );
        printf(" Block length= ");
        scanf("%d", &blk_length_in );
        printf(" \n ");

        blk_length_out = blk_length_in*ratio;
        blk_width_out = blk_width_in*ratio;
       width_out = width_in*ratio;
        printf("%d #",img_in[65]);
        extrapolate ( img_in, width_in, length_in );
        printf("%d &",img_in[65]);
        blk_in = (float *)malloc((blk_length_in+1)*(blk_width_in+1)*sizeof(float));
        if ( !blk_in )
        {
        printf("memory request failed\n");
        exit (6);
        }
        blk_out = (float *)malloc((blk_length_out+1)*(blk_width_out+1)*sizeof(float));
        if ( !blk_out )
        {
        printf("memory request failed\n");
        exit (7);
        }
```

```
        fblk_length_in = 0;
        fblk_length_out = 0;
        while ( fblk_length_in<length_in )
                {
                fblk_width_in = 0;
                fblk_width_out = 0;
                while ( fblk_width_in<width_in )
                        {
                        for ( i=0; i<=blk_length_in; i++ )
                        for ( j=0; j<=blk_width_in; j++ )
                                blk_in[i*blk_width_in+j] = (float)
                                    img_in[(i+fblk_length_in)*width_in+(j+fblk_width_in)];

                        interpolation(blk_in, blk_out, blk_length_in, blk_width_in, ratio);

                        for (i=0; i<blk_length_out; i++)
                        for (j=0; j<blk_width_out; j++)
                                img_out[(i+fblk_length_out)*width_out+(j+fblk_width_out)]
                                    = (unsigned char)blk_out[i*blk_width_out+j];

                        fblk_width_out += blk_width_out;
                        fblk_width_in += blk_width_in;
                        }
                fblk_length_out += blk_length_out;
                fblk_length_in += blk_length_in;
                }

        free ( blk_in );
        free ( blk_out );

        return;
}       /*      end of zoom   */


void extrapolate ( input_img, width_in, length_in )     /*  calling from zooming     */

unsigned char           *input_img;
int     width_in, length_in;

{       /*      extrapolate     */
```

```
        int     a,b,c;
        int     i,j;

        for ( i=0; i<length_in; i++ )
                {
                a = (int) input_img[(i+1)*(width_in+1)-4];
                b = (int) input_img[(i+1)*(width_in+1)-3];
                c = (int) input_img[(i+1)*(width_in+1)-2];
                input_img[(i+1)*(width_in+1)-1] = (unsigned char) a-3*b+3*c;
                }

        for ( j=0; j<=width_in; j++ )
                {
                a = (int) input_img[(length_in-3)*(width_in+1)+j];
                b = (int) input_img[(length_in-2)*(width_in+1)+j];
                c = (int) input_img[(length_in-1)*(width_in+1)+j];
                input_img[length_in*(width_in+1)+j] = (unsigned char) a-3*b+3*c;
                }

        return;
}       /*      end of extrapolate      */


void interpolation ( blk_in, blk_out, length_in, width_in, ratio)
                                /*  calling from zooming    */

float   *blk_in, *blk_out;
int     length_in, width_in;
int     ratio;

{       /*      interpolation   */
        int     width_out, length_out;
        int     T_flag, S_flag;
        int     i, j, Ns;
        float   *Real_in, *Real_out;
        float   *Imag_in, *Imag_out;
        float   *last_row_in, *last_row_out;
        float   *last_clm_in, *last_clm_out;

        Real_in = (float *)malloc(length_in*width_in*sizeof(float));
```

```
if ( !Real_in )
{
printf("memory request failed\n");
exit (6);
}
Real_out = (float *)malloc(length_out*width_out*sizeof(float));
if ( !Real_out )
{
printf("memory request failed\n");
exit (7);
}
Imag_in = (float *)malloc(length_in*width_in*sizeof(float));
if ( !Imag_in )
{
printf("memory request failed\n");
exit (6);
}
Imag_out = (float *)malloc(length_out*width_out*sizeof(float));
if ( !Imag_out )
{
printf("memory request failed\n");
exit (7);
}
last_row_in = (float *)malloc((width_in+1)*sizeof(float));
if ( !last_row_in )
{
printf("memory request failed\n");
exit (6);
}
last_row_out = (float *)malloc((width_out+1)*sizeof(float));
if ( !last_row_out )
{
printf("memory request failed\n");
exit (7);
}
last_clm_in = (float *)malloc((length_in+1)*sizeof(float));
if ( !last_clm_in )
{
printf("memory request failed\n");
exit (6);
}
```

```
last_clm_out = (float *)malloc((length_out+1)*sizeof(float));
if ( !last_clm_out )
{
printf("memory request failed\n");
exit (7);
}

width_out = width_in*ratio;
length_out = length_in*ratio;

T_flag=1;
L_transform ( blk_in, width_in, length_in, T_flag);

for ( i=0; i<length_in; i++ )
for ( j=0; j<width_in; j++ )
        {
        Real_in[i*width_in+j] = blk_in[i*width_in+j];
        Imag_in[i*width_in+j] = 0;
        }

S_flag=1;
spectrum ( Real_in, Imag_in, width_in, length_in, S_flag );

/*----------------------Zero Padding-----------------*/

Ns = length_in*width_in;
for ( i=0; i<length_out; i++ )
for ( j=0; j<width_out; j++ )
        {
        Real_out[i*width_out+j] = 0;
        Imag_out[i*width_out+j] = 0;
        }

for ( i=0; i<length_in/2; i++ )
for ( j=0; j<width_in/2; j++ )
        {
        Real_out[i*width_out+j] = Real_in[i*width_in+j]/Ns;
        Imag_out[i*width_out+j] = Imag_in[i*width_in+j]/Ns;
        }

for ( i=0; i<length_in/2; i++ )
```

```
for ( j=1; j<width_in/2; j++ )
        {
        Real_out[(i+1)*width_out-width_in/2+j] =
                Real_in[i*width_in+width_in/2+j]/Ns;
        Imag_out[(i+1)*width_out-width_in/2+j] =
                Imag_in[i*width_in+width_in/2+j]/Ns;
        }

for ( j=0; j<width_in/2; j++ )
for ( i=1; i<length_in/2; i++ )
        {
        Real_out[(length_out-length_in/2+i)*width_out+j] =
                Real_in[(length_in/2+i)*width_in+j]/Ns;
        Imag_out[(length_out-length_in/2+i)*width_out+j] =
                Imag_in[(length_in/2+i)*width_in+j]/Ns;
        }

for ( i=1; i<length_in/2; i++ )
for ( j=1; j<width_in/2; j++ )
        {
        Real_out[(length_out-length_in/2+i)*width_out+width_out-width_in/2+j]
                = Real_in[(length_in/2+i)*width_in+width_in/2+j]/Ns;
        Imag_out[(length_out-length_in/2+i)*width_out+width_out-width_in/2+j]
                = Imag_in[(length_in/2+i)*width_in+width_in/2+j]/Ns;
        }

for ( i=0; i<length_in/2; i++ )
        {
        Real_out[i*width_out+width_in/2] =
                0.5*Real_in[i*width_in+width_in/2]/Ns;
        Imag_out[i*width_out+width_in/2] =
                0.5*Imag_in[i*width_in+width_in/2]/Ns;
        Real_out[i*width_out+width_out-width_in/2] =
                0.5*Real_in[i*width_in+width_in/2]/Ns;
        Imag_out[i*width_out+width_out-width_in/2] =
                0.5*Imag_in[i*width_in+width_in/2]/Ns;
        }

for ( i=1; i<length_in/2; i++ )
        {
        Real_out[(length_out-length_in/2+i)*width_out+width_in/2] =
```

```
                0.5*Real_in[(length_in/2+i)*width_in+width_in/2]/Ns;
        Imag_out[(length_out-length_in/2+i)*width_out+width_in/2] =
                0.5*Imag_in[(length_in/2+i)*width_in+width_in/2]/Ns;
        Real_out[(length_out-length_in/2+i)*width_out+width_out-width_in/2] =
                0.5*Real_in[(length_in/2+i)*width_in+width_in/2]/Ns;
        Imag_out[(length_out-length_in/2+i)*width_out+width_out-width_in/2] =
                0.5*Imag_in[(length_in/2+i)*width_in+width_in/2]/Ns;
        }

for ( j=0; j<width_in/2; j++ )
        {
        Real_out[(length_in/2)*width_out+j] =
                0.5*Real_in[(length_in/2)*width_in+j]/Ns;
        Imag_out[(length_in/2)*width_out+j] =
                0.5*Imag_in[(length_in/2)*width_in+j]/Ns;
        Real_out[(length_out-length_in/2)*width_out+j] =
                0.5*Real_in[(length_in/2)*width_in+j]/Ns;
        Imag_out[(length_out-length_in/2)*width_out+j] =
                0.5*Imag_in[(length_in/2)*width_in+j]/Ns;
        }

for ( j=1; j<width_in/2; j++ )
        {
        Real_out[(length_in/2)*width_out+width_out-width_in/2+j] =
                0.5*Real_in[(length_in/2)*width_in+width_in/2+j]/Ns;
        Imag_out[(length_in/2)*width_out+width_out-width_in/2+j] =
                0.5*Imag_in[(length_in/2)*width_in+width_in/2+j]/Ns;
        Real_out[(length_out-length_in/2)*width_out+width_out-width_in/2+j] =
                0.5*Real_in[(length_in/2)*width_in+width_in/2+j]/Ns;
        Imag_out[(length_out-length_in/2)*width_out+width_out-width_in/2+j] =
                0.5*Imag_in[(length_in/2)*width_in+width_in/2+j]/Ns;
        }

Real_out[(length_in/2)*width_out+width_in/2] =
        0.25*Real_in[(length_in/2)*width_in+width_in/2]/Ns;
Imag_out[(length_in/2)*width_out+width_in/2] =
        0.25*Imag_in[(length_in/2)*width_in+width_in/2]/Ns;
Real_out[(length_in/2)*width_out+width_out-width_in/2] =
        0.25*Real_in[(length_in/2)*width_in+width_in/2]/Ns;
Imag_out[(length_in/2)*width_out+width_out-width_in/2] =
        0.25*Imag_in[(length_in/2)*width_in+width_in/2]/Ns;
```

```
Real_out[(length_out-length_in/2)*width_out+width_in/2] =
        0.25*Real_in[(length_in/2)*width_in+width_in/2]/Ns;
Imag_out[(length_out-length_in/2)*width_out+width_in/2] =
        0.25*Imag_in[(length_in/2)*width_in+width_in/2]/Ns;
Real_out[(length_out-length_in/2)*width_out+width_out-width_in/2] =
        0.25*Real_in[(length_in/2)*width_in+width_in/2]/Ns;
Imag_out[(length_out-length_in/2)*width_out+width_out-width_in/2] =
        0.25*Imag_in[(length_in/2)*width_in+width_in/2]/Ns;

/*---------------------- Change back to time domain ----------------------*/

S_flag=-1;
spectrum ( Real_out, Imag_out, width_out, length_out, S_flag );

for ( i=0; i<length_out; i++)
for ( j=0; j<width_out; j++ )
        blk_out[i*width_out+j] = Real_out[i*width_out+j];


/*------------------- Interpolate last column --------------*/

for ( i=0; i<=length_in; i++)
        last_clm_in[i] = blk_in[(i+1)*(width_in+1)-1];

One_D_interpolate ( last_clm_in, last_clm_out, length_in, length_out );

for ( i=0; i<=length_out; i++)
        blk_out[(i+1)*(width_out+1)-1] = last_clm_out[i];

/*------------------- Interpolate last row --------------*/

for ( i=0; i<=width_in; i++)
        last_row_in[i] = blk_in[length_in*(width_in+1)+i];

One_D_interpolate ( last_row_in, last_row_out, width_in, width_out );

for ( i=0; i<=width_out; i++)
        blk_out[length_out*(width_out+1)+i] = last_row_out[i];


/*---------------------- Recovery Transform -------------------*/
```

```
        T_flag=-1;
        L_transform ( blk_out, width_out, length_out, T_flag );

        free ( Real_in );
        free ( Imag_in );
        free ( Imag_out );
        free ( Real_out );

        return;
}       /*      end of interpolation   */


/*---------------------Linear Transform----------------*/

void L_transform ( blk, width,length, T_flag)  /*  calling from block    */

float   *blk;
int     width, length;
int     T_flag;

{       /*      L_transform   */
        float   p;
        float   *p_row, *p_column;
        int     i, j;

        p_row = (float *)malloc(length*sizeof(float));
        if ( !p_row )
                {
                printf("memory request failed\n");
                exit (15);
                }
        p_column = (float *)malloc(width*sizeof(float));
        if ( !p_column )
                { printf("memory request failed\n");
                exit (16);
                }

        p = (blk[0]-blk[width]-blk[length*(width+1)]+blk[(length+1)*(width+1)-1])/
            (width*length);   /*  constent coefficients  */

        p_row[0] = (blk[width]-blk[0])/width;   /*  calculate coefficients for first row   */
```

```
        p_column[0] = (blk[length*(width+1)]-blk[0])/length;
                                /*  calculate coefficients for frst column    */

        if(T_flag == -1)
                {
                for (i=1; i<width; i++)
                blk[i] = blk[i]-T_flag*p_row[0]*i;   /*  recovery first row     */
                for (i=1; i<length; i++)
                blk[i*(width+1)] = blk[i*(width+1)]-T_flag*p_column[0]*i;
                                                /*  recovery first column     */
                }

        for (i=1; i<width; i++)
                p_column[i] = (blk[length*(width+1)+i]-blk[i])/length;
                                /*  calculate coefficients for each column   */

        for (i=1; i<length; i++)
                p_row[i] = (blk[(i+1)*(width+1)-1]-blk[i*(width+1)])/width;
                                /*      calculate coefficients for each row    */

        for (i=1; i<length; i++)
        for (j=1; j<width; j++)
                blk[i*(width+1)+j] =
                        blk[i*(width+1)+j]+T_flag*(p*i*j-p_row[i]*j-p_column[j]*i);
                                /*  transform or recover each row and column */

        if (T_flag == 1)
                {
                for (i=1; i<width; i++)
                blk[i] = blk[i]-T_flag*p_row[0]*i;   /*  transform frist row    */
                for (i=1; i<length; i++)
                        blk[i*(width+1)] = blk[i*(width+1)]-T_flag*p_column[0]*i;
                                                /*  transform first column    */
                }

        free ( p_row );
        free ( p_column );

        return;
}       /*      end of linear transform   */
```

```
void spectrum ( Real, Imag, width, length, S_flag )

float   *Real, *Imag;
int     width, length;
int     S_flag;

{       /*      spectrum        */

        float   *row_real, *row_imag;
        float   *col_real, *col_imag;
        int     length_prime, width_prime;
        int     *prime;
        int     pm;
        int     i,j;

        col_real = (float *)malloc((length+1)*sizeof(float));
        if ( !col_real )
                {
                printf("memory request failed\n");
                exit (17);
                }
        col_imag = (float *)malloc((length+1)*sizeof(float));
        if ( !col_imag )
                {
                printf("memory request failed\n");
                exit (18);
                }
        row_real = (float *)malloc((width+1)*sizeof(float));
        if ( !row_real )
                {
                printf("memory request failed\n");
                exit (19);
                }
        row_imag = (float *)malloc((width+1)*sizeof(float));
        if ( !row_imag )
                {
                printf("memory request failed\n");
                exit (20);
                }

        prime = &pm;
```

```
getprime ( length, prime );
length_prime = *prime;

getprime ( width, prime );
width_prime = *prime;

for (i=0; i<length; i++)
        {
        for (j=0; j<width; j++)
                {
                row_imag[j+1] = S_flag*Imag[i*width+j];
                row_real[j+1] = Real[i*width+j];
                }

        FFT ( row_real, row_imag, width, width_prime );

        for (j=0; j<width; j++)
                {
                Real[i*width+j] = row_real[j+1];
                Imag[i*width+j] = row_imag[j+1];
                }
        }
for (j=0; j<width; j++)
        {
        for (i=0; i<length; i++)
                {
                col_real[i+1] = Real[i*width+j];
                col_imag[i+1] = Imag[i*width+j];
                }

        FFT ( col_real, col_imag, length, length_prime );

        for (i=0; i<length; i++)
                {
                Real[i*width+j] = col_real[i+1];
                Imag[i*width+j] = col_imag[i+1];
                }
        }

free ( row_real );
free ( row_imag );
```

```
        free ( col_real );
        free ( col_imag );

        return;
}       /*      end of spetrum    */

void One_D_interpolate ( In, Out, Num_in, Num_out, last_point )
float   *In, *Out;
int     Num_in, Num_out;
float   last_point;

{       /*      One_D_interpolation  */
        float   *real_in, *imag_in;
        float   *real_out, *imag_out;
        int     row_out, clm_out;
        int     prime;
        int     *pm;
        float   p1,p2;
        int     i;

        real_in = (float *)malloc((Num_in+1)*sizeof(float));
        if ( !real_in )
                {
                printf("memory request failed\n");
                exit (21);
                }
        imag_in = (float *)malloc((Num_in+1)*sizeof(float));
        if ( !imag_in )
                {
                printf("memory request failed\n");
                exit (22);
                }
        real_out = (float *)malloc((Num_out+1)*sizeof(float));
        if ( !real_out )
                {
                printf("memory request failed\n");
                exit (12);
                }
        imag_out = (float *)malloc((Num_out+1)*sizeof(float));
        if ( !imag_out )
                {
```

```
        printf("memory request failed\n");
        exit (23);
        }

pm = &prime;

getprime ( Num_in, pm );
prime = *pm;

p1 = (In[Num_in]-In[0])/Num_in;
p2 = (In[Num_in]-In[0])/Num_out;

for (i=0; i<Num_in; i++)
        real_in[i+1] = In[i]-p1*i;

for (i=0; i<=Num_in; i++)
        imag_in[i] = 0;

FFT(real_in, imag_in, Num_in, prime);

for (i=1; i<=Num_out; i++)
        {
        real_out[i] = 0;
        imag_out[i] = 0;
        }
for (i=1; i<=Num_in/2; i++)
        {
        real_out[i] = real_in[i]/Num_in;
        imag_out[i] = imag_in[i]/Num_in;
        }
for (i=2; i<=Num_in/2; i++)
        {
        real_out[Num_out-Num_in/2+i] = real_in[Num_in/2+i]/Num_in;
        imag_out[Num_out-Num_in/2+i] = imag_in[Num_in/2+i]/Num_in;
        }
real_out[Num_out-Num_in/2+1] = 0.5*real_in[Num_in/2+1]/Num_in;
real_out[Num_in/2+1] = 0.5*real_in[Num_in/2+1]/Num_in;
imag_out[Num_out-Num_in/2+1] = 0.5*imag_in[Num_in/2+1]/Num_in;
imag_out[Num_in/2+1] = 0.5*imag_in[Num_in/2+1]/Num_in;

getprime(Num_out, pm);
```

```
        prime = *pm;

        FFT(imag_out, real_out, Num_out, prime);

        for (i=0; i<Num_out; i++)
                Out[i] = real_out[i+1]+p2*i;

        free ( real_in );
        free ( imag_in );
        free ( real_out );
        free ( imag_out );

        return;
}       /*      end of One_D_interpolation */


void getprime ( num, prime )

int     num;
int     *prime;

{       /*      getprime        */

        *prime = 0;
        while (num != 1)
                {
                num=num/2;
                (*prime)++;
                }
        return;
}       /*      end of getprime   */


void FFT ( Re, Im, Nz, M )
float   *Re, *Im;
int     M, Nz;

{       /*      FFT    */
        int     i, j, k, I0;
        int     No1, No2, N4, IS, ID, I1, I2, I3;
        float   E, A, A3, R1, R2, S1, S2, S3;
```

```
float   CC1, CC3, SS1, SS3, XT;


No2=2*Nz;
for (k=1; k<M; k++)
        {
        No2 = No2/2;
        N4 = No2/4;
        E = 2*pi/No2;
        A = 0;
        for (j=1; j<=N4; j++)
                {
                A3 = 3*A;
                CC1 = cos(A);
                SS1 = sin(A);
                CC3 = cos(A3);
                SS3 = sin(A3);
                A = j*E;
                IS = j;
                ID = 2*No2;
                while ( IS<Nz )
                        {
                        for (I0=IS; I0<Nz; I0=I0+ID)
                            {
                            I1 = I0+N4;
                            I2 = I1+N4;
                            I3 = I2+N4;

                            R1 = Re[I0]-Re[I2];
                            Re[I0] = Re[I0]+Re[I2];
                            R2 = Re[I1]-Re[I3];
                            Re[I1] = Re[I1]+Re[I3];
                            S1 = Im[I0]-Im[I2];
                            Im[I0] = Im[I0]+Im[I2];
                            S2 = Im[I1]-Im[I3];
                            Im[I1] = Im[I1]+Im[I3];

                            S3 = R1-S2;
                            R1 = R1+S2;
                            S2 = R2-S1;
                            R2 = R2+S1;
```

```
                        Re[I2] = R1*CC1-S2*SS1;
                        Im[I2] = -S2*CC1-R1*SS1;
                        Re[I3] = S3*CC3+R2*SS3;
                        Im[I3] = R2*CC3-S3*SS3;
                        }
                    IS = 2*ID-No2+j;
                    ID = 4*ID;
                    }
                }
        }

/*-------------LAST STAGE, LENGTH-2 BUTTERFLY-----------*/

IS = 1;
ID = 4;
while (IS<Nz)
        {
        for (I0=IS; I0<=Nz; I0=I0+ID)
                {
                I1 = I0+1;
                R1 = Re[I0];
                Re[I0] = R1+Re[I1];
                Re[I1] = R1-Re[I1];
                R1 = Im[I0];
                Im[I0] = R1+Im[I1];
                Im[I1] = R1-Im[I1];
                }
        IS = 2*ID-1;
        ID = 4*ID;
        }

/*----------------BIT REVERSE COUNTER-------------------*/

j = 1;
No1 = Nz-1;
for (i=1; i<=No1; i++)
        {
        if (i<j)
                {
                XT = Re[j];
                Re[j] = Re[i];
```

```
                    Re[i] = XT;
                    XT = Im[j];
                    Im[j] = Im[i];
                    Im[i] = XT;
                    }
            k = Nz/2;
            while(k<j)
                    {
                    j = j-k;
                    k = k/2;
                    }
            j = j+k;
            }
    return;
}   /*   end of FFT   */
```

VITA AUCTORIS

NAME: Zhiwei (Jerry) Wang

PLACE OF BIRTH: P.R.China

YEAR OF BIRTH: 1966

EDUCATION:

B.Eng.
Department of Computer and Telecommunications
Beijing University of Posts and Telecommunications
Beijing, P.R. China

M.A.Sc.
Department of Electrical Engineering
University of Windsor
Windsor, Ontario
Canada